



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Macam dari perangkat lunak dengan komponen-komponennya. Aplikasi yang dihasilkan dengan perangkat lunak.

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

Kode MK
87011

Disusun Oleh
Tim Dosen

01

Abstract

Membahas mengenai macam, komponen dan aplikasi perangkat lunak

Kompetensi

Mahasiswa dapat mengetahui komponen-komponen dari perangkat lunak dan aplikasi yang dihasilkannya.

Macam dari Perangkat Lunak

Perangkat Lunak atau software computer dapat dikelompokkan dalam dua kelompok, yakni :

1. Perangkat lunak system (Software system)
2. Perangkat lunak Aplikasi (Software aplikasi)

Perangkat lunak system dibedakan menjadi 3, yaitu :

1. System operasi
2. Bahasa pemrograman
3. Utilitas

Sedangkan Perangkat lunak aplikasi dibedakan menjadi :

- Aplikasi perkantoran
- Aplikasi multimedia
- Aplikasi internet dan jaringan
- Aplikasi khusus

Macam-macam Perangkat Lunak Sistem Operasi :

1. DOS (Disc Operating System); system operasi generasi awal yang dirancang untuk computer tunggal Personal Computer (PC).
2. Unix; Sistem operasi berbasis jaringan, merupakan system operasi tertua, dikeluarkan tahun 1960. Unix pertama kali digunakan oleh computer jenis IBM, HP dan Sun Solaris.

Bagian-bagian Unix yaitu :

1. Unix
2. Open BSD
3. FreeBSD

Windows 95/ windows 98/ windows Me/ windows XP

Sistem operasi buatan Microsoft ini sangat populer karena tampilannya yang user friendly dengan menggunakan pendekatan GUI. Kelebihan windows dibandingkan dengan system operasi lainnya yaitu :

- ✓ Multitasking; kemampuan yang memungkinkan penggunaan sejumlah program dalam waktu bersamaan.
- ✓ Mendukung system kerja team atau workgroup dalam suatu jaringan.

✓
Macintosh;
dikeluarkan pada Januari 1984

Linux; diperkenalkan pertama kali oleh Linus Torvalds tahun 1991. Linux menjadi pesaing utama system operasi windows karena mempunyai keunggulan yang sama yakni mendukung multitasking, user friendly, serta workgroup.

Perangkat Lunak Bahasa Pemrograman

Terdapat empat kelompok generasi bahasa pemrograman :

1. Generasi Pertama (Bahasa Mesin); pemrograman menggunakan bahasa mesin yang diwakili oleh bilangan biner 0 dan 1.
2. Generasi Kedua (Bahasa Assembly); pemrograman menggunakan perintah kata yang pendek.
3. Generasi Ketiga (Bahasa tingkat tinggi); pemrograman dengan bahasa yang procedural tertata dengan baik.
4. Generasi Keempat (Bahasa pemrograman yang berorientasi pada object)

Aplikasi yang dihasilkan dengan perangkat lunak

Perangkat Lunak Utilitas yaitu perangkat lunak yang ditujukan untuk menunjang fungsionalitas perangkat lunak system operasi. Contoh untuk melakukan kompresi data pada harddisk atau media penyimpanan lain, dapat dilakukan melalui perangkat lunak WinZip. Contoh lainnya apabila untuk menangkal virus diperlukan perangkat lunak antivirus.

Aplikasi Perkantoran

Aplikasi perkantoran yaitu perangkat lunak yang ditujukan untuk membantu tugas-tugas dalam dunia perkantoran.

Yang termasuk Aplikasi perkantoran , adalah :

1. Spreadsheet; Yang sering dipergunakan untuk menyelesaikan pekerjaan kantor khususnya dibidang hitung menghitung. Yang paling banyak digunakan adalah Microsoft excel.
2. Word processor; adalah aplikasi pengolah kata. Nama programnya adalah Microsoft word.
3. Program Presentasi; Sebuah aplikasi untuk membuat presentasi. Nama program yang paling populer yaitu Microsoft Power Point.

4. Data base manajemen system; adalah perangkat lunak untuk melaksanakan manajemen data. Yang paling banyak digunakan dalam office adalah Microsoft visual basic.

Aplikasi Multimedia

Aplikasi yang mendukung teknologi multimedia, seperti teks, suara, gambar, film. Macam-macam perangkat lunak multimedia :

1. Corel Draw dan adobe photoshop ; aplikasi untuk membuat desain gambar dan foto.
2. RealPlayer, Winamp, Windows media player; adalah program untuk memutar music dan film.
3. Adobe premiere; perangkat lunak untuk membuat dan mengedit film.
4. Macromedia flash MX; program untuk membuat berbagai animasi.

Perangkat Lunak aplikasi internet dan jaringan

Perangkat Lunak aplikasi internet dan jaringan yaitu perangkat lunak yang digunakan untuk mendukung pemanfaatn internet dan jaringan. Beberapa perngkat lunak yang terkait dengan internet dan jaringan antara lain :

1. Web browser; adalah program untuk mengakses informasi internet, contohnya Internet Explorer, Opera, Mozilla firefox.
2. E-mail software; perangkat lunak yang menyediakan fasilitas untuk berkomunikasi. Contohnya Microsoft outlook.
3. ICQ; merupakan singkatan “I Seek You” adalah sebuah program untuk berchatting.

Perangkat Lunak aplikasi Khusus

Adalah perangkat lunak yang ditujukan pada bidang-bidang spesifik, contohnya:

1. Program SPSS; untuk analisis data statistic.
2. Program Matematika dan MAPLE, perangkat lunak pada bidang kajian matematika.
3. Program AutoCad; adalah program untuk desain pada ilmu teknik arsitektur.
4. Program MYOB, DEA, GL; untuk keperluan akuntansi perusahaan.

Daftar Pustaka

- Software Engineering Ian Sommerville

- Software Engineering Roger S.Pressman







MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Arti dan definisi dari perangkat lunak. Jenis-jenis perangkat lunak. Pentingnya rekayasa perangkat lunak.

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

02

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Membahas mengenai definisi, komponen dan aplikasi perangkat lunak

Kompetensi

Mahasiswa dapat mengetahui komponen-komponen dari perangkat lunak dan aplikasi yang dihasilkannya.

Sejarah Rekayasa Perangkat Lunak

Rekayasa perangkat lunak telah berkembang sejak pertama kali diciptakan pada tahun 1940-an hingga kini. Fokus utama pengembangannya adalah untuk mengembangkan praktek dan teknologi untuk meningkatkan produktivitas para praktisi pengembangan perangkat lunak dan kualitas aplikasi yang dapat digunakan oleh pemakai.

1945 - 1965: Awal

Istilah software engineering digunakan pertama kali pada akhir 1950-an dan awal 1960-an. Saat itu, masih terdapat debat tajam mengenai aspek engineering dari pengembangan perangkat lunak.

Pada tahun 1968 dan 1969, komite sains NATO mensponsori dua konferensi tentang rekayasa perangkat lunak, yang memberikan dampak kuat terhadap perkembangan rekayasa perangkat lunak. Banyak yang menganggap bahwa dua konferensi inilah yang menandai awal resmi profesi rekayasa perangkat lunak.

1965 - 1985: krisis perangkat lunak

Pada tahun 1960-an hingga 1980-an, banyak masalah yang ditemukan para praktisi pengembangan perangkat lunak. Banyak proyek yang gagal, hingga masa ini disebut sebagai krisis perangkat lunak. Kasus kegagalan pengembangan perangkat lunak terjadi mulai dari proyek yang melebihi anggaran, hingga kasus yang mengakibatkan kerusakan fisik dan kematian. Salah satu kasus yang terkenal antara lain meledaknya roket Ariane akibat kegagalan perangkat lunak.

1985 - kini: tidak ada senjata pamungkas

Selama bertahun-tahun, para peneliti memfokuskan usahanya untuk menemukan teknik jitu untuk memecahkan masalah krisis perangkat lunak. Berbagai teknik, metode, alat, proses diciptakan dan diklaim sebagai senjata pamungkas untuk memecahkan kasus ini. Mulai dari pemrograman terstruktur, pemrograman berorientasi object, perangkat pembantu pengembangan perangkat lunak (CASE tools), berbagai standar, UML hingga metode formal diagungkan sebagai senjata pamungkas untuk menghasilkan software yang benar, sesuai anggaran dan tepat waktu.

Pada tahun 1987, Fred Brooks menulis artikel No Silver Bullet, yang berproposisi bahwa tidak ada satu teknologi atau praktek yang sanggup mencapai 10 kali lipat perbaikan dalam produktivitas pengembangan perangkat lunak dalam tempo 10 tahun. Sebagian berpendapat, no silver bullet berarti profesi rekayasa perangkat lunak dianggap telah gagal. Namun sebagian yang lain justru beranggapan, hal ini menandakan bahwa bidang profesi rekayasa perangkat lunak telah cukup matang, karena dalam bidang profesi lainnya pun, tidak ada teknik pamungkas yang dapat digunakan dalam berbagai kondisi

Definisi Rekayasa Perangkat Lunak

Berikut adalah beberapa definisi dari istilah Rekayasa Perangkat Lunak :

- Menurut Stephen R.Schach
sebuah disiplin dimana dalam menghasilkan perangkat lunak bebas dari kesalahan dan dalam pengiriman anggaran tepat waktu serta memuaskan keinginan pemakai.
- Menurut Fritz Bauer
penetapan dan penggunaan prinsip rekayasa dalam rangka memperoleh perangkat lunak yang dapat dipercaya dan dapat bekerja secara efisien pada mesin nyata.
- Menurut IEEE 610.12
sebuah studi pendekatan dan aplikasi secara sistematis, disiplin pengembangan operasi dan pemeliharaan PL yang kesemuanya itu merupakan aplikasi rekayasa yang berkaitan dengan PL.
- Menurut Wikipedia
"Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software". Secara ringkasnya adalah bahwa SE mencakup pembuatan, pengembangan, dan pemeliharaan suatu software.

Pembuatan meliputi bagaimana suatu software dibuat mulai dari user requirements, spesifikasi, desain, testing, dokumentasi (misal berupa manual pembuatan program), dan sebagainya. Sedangkan pengembangan adalah untuk menambah fitur-fitur baru yang belum ada pada versi sebelumnya. Pemeliharaan digunakan untuk memperbaiki bugs atau errors yang tidak ketahuan ketika dalam tahap pembuatan. Pemeliharaan ini biasanya dapat berupa Service Pack, dan sebagainya.

Definisi Perangkat Lunak

Beberapa definisi dari perangkat lunak antara lain :

- Perangkat lunak adalah program komputer ditambah konfigurasi data dan file serta ditambahkan juga dokumentasi.
- Menurut Ian Sommerville :
"Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market." Program komputer dan dokumentasi yang terkait. Produk Software dapat dikembangkan untuk pelanggan tertentu atau mungkin dikembangkan untuk umum.
- Menurut Pressman dalam bukunya Software Engineering A Practitioner's Approach, perangkat lunak didefinisikan lebih rinci lagi yaitu sebagai:
 - instruksi-instruksi yang jika dieksekusi akan memberikan layanan-layanan atau fungsi seperti yang diinginkan
 - struktur data yang memungkinkan program untuk memanipulasi informasi secara proporsional
 - dokumen-dokumen yang menggambarkan operasi dan kegunaan program
- Menurut IEEE (Standard Glossary of Software Engineering Terminology, 1990), perangkat lunak adalah program komputer, prosedur, dan dokumentasi serta data yang terkait dengan pengoperasian sistem komputer.

Peranan Perangkat Lunak

Software saat ini memegang dua peran. Software sebagai sebuah produk dan, dalam waktu yang bersamaan, juga sebagai sarana untuk menghasilkan sebuah produk. Sebagai sebuah produk, software memberikan suatu kemampuan menghitung yang disatukan ke dalam hardware komputer, atau yang lebih luas lagi, pada jaringan komputer yang bisa diakses oleh hardware lokal. Software bisa berada melekat didalam sebuah telepon selular atau beroperasi didalam sebuah komputer mainframe, yang bekerja untuk merubah data menjadi informasi, memproduksi, mengatur, mendapatkan, mengubah, mempertunjukan, atau mengirimkan informasi yang sederhana, seperti sebuah bit tunggal atau, barangkali sekompleks pertunjukan dengan multimedia. Sebagai sarana yang digunakan untuk menghasilkan suatu produk, software bertindak sebagai basis untuk mengontrol komputer (sistem operasi), komunikasi informasi (jaringan), dan pembuat serta pengontrol dari program yang lainnya (software sebagai alat bantu (tools) dan lingkungan (environments)).

Software menghantar informasi, produk paling penting yang dihasilkan untuk kita, mengubah data pribadi (seperti misalnya transaksi keuangan pribadi) sehingga data itu bisa lebih berarti dalam konteks lokal; software juga mengelola informasi bisnis menjadikannya lebih bisa bersaing; juga menjadi penyedia pintu masuk ke jaringan informasi dunia(misal internet) dan yang bisa memenuhi permintaan informasi dalam berbagai bentuk.

Peran software komputer cukup banyak mengalami perubahan sepanjang kurun waktu tidak kurang dari 50 tahun. Dengan peningkatan kemampuan hardware yang dramatis, perubahan yang terjadi lebih dalam lagi dari segi arsitektur komputer, kecepatan, peningkatan kapasitas memori dan penyimpanan, dan dengan beragamnya variasi pilihan input dan output menjadikan sistem berbasis komputer semakin lengkap dan kompleks. Kehandalan dan kompleksitas bisa menimbulkan kekaguman ketika suatu sistem yang berjalan dengan baik dan sukses, tetapi dia juga bisa menimbulkan masalah yang besar manakala mereka harus membangun sebuah sistem yang rumit.

Perkembangan Perangkat Lunak

- Tahun-Tahun Awal (1950 - 1965)
 - Orientasi batch -> update data pada periode tertentu
 - Distribusi terbatas
 - PL dibuat menurut pesanan
- Era Kedua (1965 - 1975)
 - Multiuser -> ada pembagian hak akses, contoh : manager, karyawan
 - Real time -> update data langsung ketika ada perubahan
 - Database -> karena real time
 - Software produk
- Era Ketiga (1975 - 1989)
 - Sistem terdistribusi
 - Embedded Intelligence
 - Hardware biaya rendah -> kalau dulu mahal karena ukurannya sangat besar
- Era Keempat (1989 - sekarang)
 - Sistem desktop bertenaga kuat

- Teknologi berorientasi objek (Object Oriented) -> kalau ada komponen rusak, tidak perlu membeli PL baru, cukup membeli komponen
- Sistem pakar -> bertindak seperti pakar
- Jaringan syaraf tiruan
- Komputasi Paralel
- Komputasi Jaringan

Karakteristik Perangkat Lunak

Untuk menambah pemahaman mengenai software (dan mengerti sepenuhnya Software Engineering), sangatlah penting untuk memeriksa ciri-ciri software yang membuat beda dengan barang-barang yang dibuat orang lainnya. Sewaktu software dibangun, proses kreasi manusia (analisa, perancangan, pembangunan, percobaan/tes) pada akhirnya diwujudkan kedalam bentuk fisik. Jika kita bangun sebuah komputer yang baru, mula-mula kita buat sketsa, gambar rancangan formal, dan dari bentuk dasar rancangan fisik tersebut, kemudian berkembang menjadi bentuk produk secara fisik (chips, circuit board, power supply, dll).

Software adalah suatu kerangka berpikir atau logika bukan seperti elemen yang dapat dilihat secara fisik. Oleh karena itu software mempunyai ciri-ciri yang berbeda dibanding dengan perangkat keras (hardware) sebagai berikut :

1. *Software is developed or engineered, it is not manufactured in the classical sense.* (Perangkat Lunak dibangun dan dikembangkan, tidak dibuat dalam bentuk klasik.)

Perangkat lunak adalah suatu produk yang lebih menekankan pada kegiatan rekayasa (engineering) dibandingkan kegiatan manufacturing (rancang bangun di pabrik). Dalam pembuatan perangkat lunak kualitas yang tinggi dicapai melalui perancangan yang baik, tetapi dalam fase perangkat keras, selalu saja ditemukan masalah kualitas yang tidak mudah untuk disesuaikan dengan perangkat lunak. Biaya untuk perangkat lunak dikonsentrasikan pada pengembangan. Hal ini berarti proyek perangkat lunak tidak dapat diatur seperti pengaturan pada proyek pemanufacturan.

2. *Software doesn't "wear out".* (Perangkat lunak tidak pernah usang)

Perangkat lunak tidak rentan terhadap pengaruh lingkungan yang merusak yang mengakibatkan perangkat keras menjadi usang. Kesalahan-kesalahan yang tidak dapat ditemukan menyebabkan tingkat kegagalan menjadi sangat tinggi pada awal hidup program. Tetapi hal itu dapat diperbaiki (diharapkan tidak ditemukan lagi kesalahan lain) sehingga kurva menjadi mendatar.

3. *Although the industry is moving toward component based construction most software continues to be custom built* (Meskipun industri saat ini menuju pada pembangunan dengan component based namun sebagian besar software masih dibangun secara custom built.)

Kini paradigma baru mulai dikembangkan, yaitu konsep reuseability. Komponen software didisain dan diimplementasikan agar dapat digunakan kembali pada program yang berlainan.

Karakteristik Perangkat Lunak Berkualitas

Perangkat lunak yang dikatakan bagus atau berkualitas memiliki karakteristik sebagai berikut :

- Maintainability

adalah tingkat kemudahan perangkat lunak tersebut dalam mengakomodasi perubahan-perubahan

Contoh : AVG merupakan software antivirus yang memiliki tingkat maintainability cukup tinggi. AVG dapat mengupdate dirinya sendiri selama komputer memiliki koneksi dengan internet atau dengan mendownload update terbarunya di situs AVG. update tersedia tiap hari dan merupakan salah satu kelebihan avg dibanding dengan beberapa antivirus lain dalam hal maintainability.

- **Dependability**
adalah ketidakbergantungan perangkat lunak dengan elemen-elemen sistem lainnya atau sistem secara keseluruhan. Artinya kegagalan elemen lain tidak mempengaruhi performansi perangkat lunak
Contoh : AVG bergantung pada sistem operasi dan Selama Operating Sistem tidak ada masalah maka AVG tidak akan bermasalah
- **Efficiency**
Menyangkut waktu eksekusi. Waktu eksekusi cukup singkat, dan saat melakukan scanning membutuhkan waktu yang lebih singkat bila dibandingkan dengan beberapa antivirus lain
- **Usability**
adalah atribut yang menunjukkan tingkat kemudahan pengoperasian perangkat lunak

Contoh : awalnya kita membutuhkan waktu agar terbiasa dengan Interface AVG. AVG Control-Center adalah komponen utama untuk mengontrol system AVG, dan berjalan tiap kali user melakukan login. Dengan menggunakan AVGCC settingan sistem AVG dapat diedit dan kita dapat monitoring status dari tiap komponen individual seperti status updatenya.

Jenis – jenis Perangkat Lunak

1. System software

Melayani program-program yang lain, contoh :kompiler, editor, prosesor telekomunikasi, sistem operasi, driver. Areanya ditandai dengan eratnya interaksi dengan hardware komputer, penggunaan oleh banyak user, operasi konkuren yang membutuhkan penjadwalan, tukar-menukar sumber dan pengaturan proses yang canggih serta struktur data yang kompleks dan interface eksternal yang ganda.

2. Application Software

Program stand alone yang dimanfaatkan untuk menyelesaikan kebutuhan spesifik dari bisnis. Aplikasi dalam area proses bisnis maupun data teknis yang ada di dalamnya digunakan untuk memfasilitasi operasional bisnis dan digunakan untuk memfasilitasi pengambilan keputusan ditingkat manajemen maupun teknis. Selain sebagai pengolah data konvensional, software aplikasi juga digunakan untuk mengontrol fungsi bisnis secara real time (misal : pemrosesan transaksi point of sale, kontrol pemrosesan manufaktur secara real time).

3. Engineering / scientific software

Dimulai dengan algoritma numerik (number crunching). Memiliki jangkauan aplikasi mulai astronomi sampai vulkanologi, analisis otomatis sampai dinamika orbit pesawat

ruang angkasa, dan biologi molekular sampai pabrik yang sudah diotomatisasi. Namun aplikasi baru dalam area teknik atau ilmu pengetahuan sedang bergerak menjauhi algoritma numerik yang konvensional.

4. Embedded software

Ada dalam ROM, digunakan untuk mengontrol hasil serta sistem untuk keperluan konsumen dan pasar industri. Dapat melakukan fungsi terbatas serta fungsi esoterik (contoh : key pad control microwave yang bisa mematikan otomatis sesuai waktu) atau memberikan kemampuan kontrol dan fungsi penting (contoh : fungsi digital dalam sebuah automobil seperti kontrol bahan bakar, autopilot, penampilan dashboard, sistem rem).

5. Product line software

Dirancang agar dapat memiliki kemampuan khusus yang diperuntukkan bagi pelanggan yang berbeda. Product line software dapat difokuskan pada pasar terbatas (misal : pengontrolan persediaan produk), atau berfokus pada pasar konsumen massal/umum (misal : pengolah kata, spreadsheet, pengolahan database, komputer grafik, multimedia, entertainment serta aplikasi keuangan personal dan bisnis).

6. Aplikasi web

Disebut "WebApps". Dalam bentuk sederhana, WebApps merupakan kumpulan link files hypertext yang mempresentasikan informasi menggunakan text dan grafikal. Contoh : web 2.0.

7. AI (Artificial Intelligence) software

Menggunakan algoritma non-numerik untuk menyelesaikan masalah kompleks yang tidak sesuai untuk perhitungan maupun analisis secara langsung. Contoh : sistem pakar, aplikasi dengan jaringan syaraf tiruan, image dan suara, pembuktian teorema, permainan game.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Macam dari siklus hidup perangkat lunak (SWDLC/Software Development Life Cycle)

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

03

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

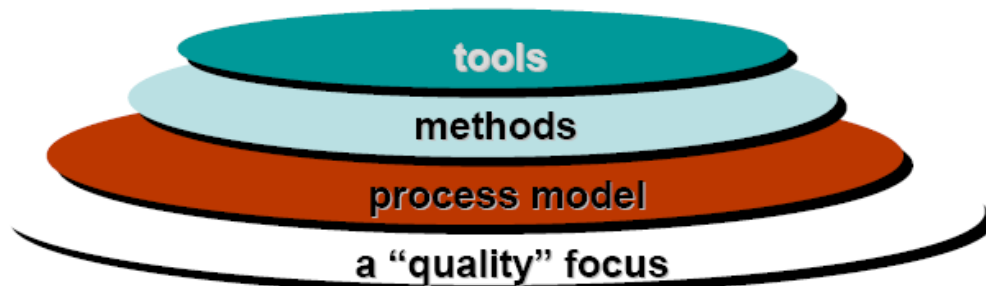
Membahas mengenai alur kehidupan yang membentuk perangkat lunak.

Kompetensi

Mahasiswa dapat memahami siklus hidup dari perangkat lunak dengan mengetahui dari segi software.

Teknologi Berlapis Pengembangan Perangkat Lunak

Proses perangkat lunak adalah sebuah kerangka kerja untuk membangun perangkat lunak yang berkualitas tinggi. Gambar 2.5 dibawah menunjukkan lapisan teknologi pada rekayasa Perangkat lunak.



Lapisan-lapisan Rekayasa Perangkat Lunak

- Dari Gambar tersebut dapat dilihat bahwa tujuan utama rekayasa perangkat lunak adalah pencapaian kualitas ("Quality Focus"). Kualitas ini diterjemahkan ke dalam ukuran-ukuran (metrics), meliputi maintainability, dependability, usability, dan efficiency yang sudah diterangkan di atas.
- Proses : mendefinisikan kerangka kerja (frame work) , sehingga pembangunan perangkat lunak dapat dilakukan secara sistematis.
- Metode : mendefinisikan bagaimana perangkat lunak dibangun, meliputi metode-metode yang digunakan dalam melakukan analisis kebutuhan, perancangan, implementasi dan pengujian. Sebagai contoh : metode terstruktur, metode berorientasi objek, dan lain-lain.
- Alat Bantu : perangkat yang bersifat otomatis maupun semi otomatis yang berfungsi mendukung tiap tahap pembangunan perangkat lunak. Contoh : CASE, CAD, dan lain-lain.
-

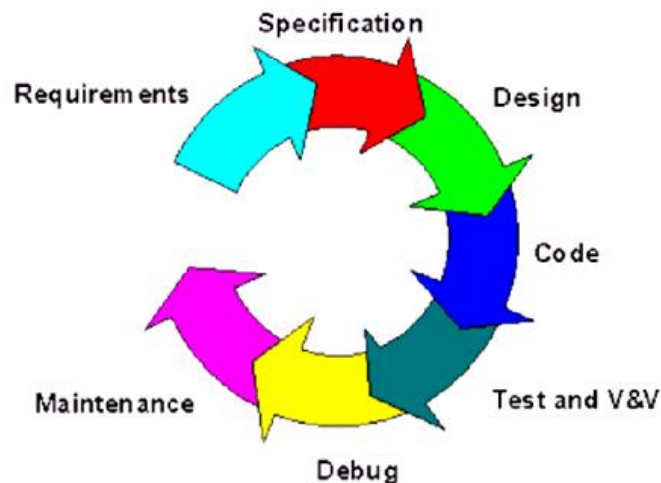
Definisi *Software Process*

Software process dapat didefinisikan sebagai berikut :

- Merupakan suatu deskripsi proses yang dijadikan panduan kerja bagi para software engineer dengan memetakan peran dan tanggung jawab mereka masing-masing.
- Merupakan sekumpulan aktifitas yang ditujukan untuk melakukan pengembangan maupun evolusi software.
- Merupakan suatu urutan langkah yang diperlukan dalam pengembangan atau pemeliharaan software.
- Merupakan kerangka teknis dan manajemen untuk menerapkan metode, alat bantu(tool) serta komponen SDM dalam pengerjaan software.
- (Menurut Ian Sommerville) : merupakan sekumpulan proses dan hasil yang terkait dengan proses tersebut dalam rangka pengembangan produk software.

Aktifitas Umum Dalam Software Process

Untuk mengembangkan perangkat lunak secara memadai, proses pengembangan perangkat lunak harus didefinisikan terlebih dahulu. Usaha yang berhubungan dengan rekayasa perangkat lunak dapat dikategorikan ke dalam beberapa aktifitas dengan tanpa mempedulikan area aplikasi, ukuran proyek atau kompleksitasnya. Berikut adalah aktifitas umum yang terdapat dalam proses pengembangan perangkat lunak :



1. **Requirement**
Merupakan aktifitas dimana didefinisikan mengenai “apa” (what) yang akan dibangun terkait dengan produk perangkat lunak yang akan dihasilkan. Kebutuhan dari persepsi pelanggan (requirement) didefinisikan dan disepakati. Dari aktifitas ini akan diperoleh pernyataan global mengenai kegunaan sistem serta ketersediaan sumber daya yang akan mendukung pembangunan sistem seperti : kebutuhan sumber daya waktu, biaya dan tenaga (manusia).
2. **Specification**
Merupakan aktifitas dimana kebutuhan pelanggan (requirement) yang telah ditetapkan ditransformasikan ke dalam kebutuhan sistem. Dari aktifitas ini akan diperoleh spesifikasi detil mengenai produk perangkat lunak yang akan dibangun antara lain seputar fungsionalitasnya (mengidentifikasi informasi apa yang akan diproses, fungsi dan unjuk kerja apa yang dibutuhkan, tingkah laku sistem seperti apa yang diharapkan), kebutuhan perangkat keras dan perangkat lunak pendukung dalam pembangunannya, dan lain-lain.
3. **Design**
Merupakan aktifitas dimana hasil analisis kebutuhan dan spesifikasi sistem dibentuk dalam suatu model. Pengembang harus mendefinisikan bagaimana data dikonstruksikan, bagaimana fungsi-fungsi diimplementasikan sebagai sebuah arsitektur perangkat lunak, bagaimana detail prosedur akan diimplementasikan, bagaimana interface ditandai (dikarakterisasi). Dalam membuat pemodelan, pengembang dapat menggambarkan pemodelan berdasarkan perilaku sistem ataupun secara struktural. Dari aktifitas ini akan diperoleh penggambaran sistem dalam bentuk model semacam use case diagram, data flow diagram, sequence diagram, entity relationship diagram, dan lain-lain.
4. **Code**
Merupakan aktifitas dimana hasil rancangan (model) dari tahapan sebelumnya diterjemahkan dalam bentuk coding program pada sebuah bahasa pemrograman.

5. Test (verification & validation)
Setelah rancangan diterjemahkan ke dalam bentuk coding program selanjutnya akan dilakukan proses pengujian untuk memastikan apakah aplikasi yang dibangun sudah sesuai dengan spesifikasi dan kebutuhan pelanggan yang ditetapkan awal ditetapkan.
6. Debug
Merupakan tahapan dimana akan dilakukan proses perbaikan yang diperlukan apabila pada fase pengujian masih ditemukan adanya kesalahan.
7. Maintenance
Aktifitas ini berfokus pada perubahan (change), yang dihubungkan dengan koreksi kesalahan, penyesuaian yang dibutuhkan ketika lingkungan perangkat lunak berkembang, serta perubahan sehubungan dengan perkembangan yang disebabkan oleh perubahan kebutuhan pelanggan. Fase pemeliharaan mengaplikasikan lagi langkah-langkah pada fase definisi dan fase pengembangan, tetapi semuanya tetap bergantung pada konteks perangkat lunak yang ada. Ada empat tipe perubahan yang terjadi selama masa fase pengembangan yaitu :
 - a. Koreksi (corrective)
Meskipun dengan jaminan kualitas yang terbaik, seperti halnya pelanggan akan tetap menemukan cacat pada perangkat lunak. Pemeliharaan korektif mengubah perangkat lunak, membenarkan cacat atau rusak.
 - b. Adaptasi (adaptive)
Dari waktu ke waktu, lingkungan original (contohnya CPU, sistem operasi, aturan-aturan bisnis, karakterisasi produk eksternal) dimana perangkat lunak dikembangkan akan terus berubah. Pemeliharaan adaptif menghasilkan modifikasi kepada perangkat lunak untuk mengakomodasi perubahan pada kebutuhan fungsional original.
 - c. Pengembangan (perfective)
Ketika perangkat lunak dipakai, pelanggan akan mengenali fungsi-sungsi tambahan yang memberi mereka keuntungan. Pemeliharaan perfektif memperluas perangkat lunak sehingga melampaui kebutuhan fungsi originalnya.
 - d. Pencegahan (preventive)
Keadaan perangkat lunak semakin memburuk sehubungan dengan waktu, dan karena itu preventive maintenance yang sering juga disebut software engineering (rekayasa perangkat lunak), harus dilakukan untuk memungkinkan perangkat lunak melayani kebutuhan para pemakainya. Pada dasarnya preventive maintenance melakukan perubahan pada program komputer sehingga bisa menjadi lebih mudah untuk dikoreksi, disesuaikan dan dikembangkan.

Aktifitas dasar dalam software process menurut Ian Sommerville antara lain :

1. Specification
Aktifitas dimana pelanggan dan pengembang membuat definisi mengenai software yang akan dibangun dan juga batasan pada pengembangannya.
2. Development
Aktifitas dimana software diproduksi (didisain dan diprogram)
3. Validation
Aktifitas dimana dilakukan proses pengecekan apakah software yang dibangun sudah sesuai dengan keinginan pelanggan.
4. Evolution
Aktifitas dimana dilakukan perubahan/modifikasi terhadap software untuk diadaptasikan terhadap perubahan kebutuhan dari pelanggan.

Aktifitas atau langkah-langkah yang dijabarkan dalam pengembangan perangkat lunak tersebut harus diimbangi dengan sejumlah aktifitas pelindung (*umbrella activities*). Kegiatan-kegiatan khusus di dalam kategori ini menyangkut :

- Manajemen proyek PL : melindungi agar PL yang ada hasilnya bagus
 - Formal technical review, contoh : menemui user dan mengecek kebutuhannya untuk analisis. Jika tidak dilakukan, nanti kita buat PL yang sesuai pikiran kita, bukan sesuai dengan kebutuhan user.
 - Software quality assurance (jaminan kualitas PL) : langkah supaya PL berkualitas
 - Manajemen konfigurasi PL : bagaimana PL bisa dikonfigurasi atau dibuat
 - Pembuatan dan penyiapan dokumen : sebagai senjata jika user tiba-tiba meminta tambahan fungsi PL
 - Reusability management (manajemen reusabilitas): komponen PL bisa dipakai ulang
 - Measurement (pengukuran) : harus ada dalam setiap tahap
 - Risk Management (manajemen resiko) : risiko harus diantisipasi supaya tidak gagal
- Aktifitas pelindung diaplikasikan ke seluruh proses perangkat lunak

Karakteristik *Software Process* Yang Baik

- Understandability
Proses secara eksplisit didefinisikan sehingga mudah dipahami bagi siapapun yang terlibat di dalam proses pengembangan.
- Visibility
Aktifitas proses memberi hasil yang jelas sehingga kemajuan proses dapat terlihat dari luar pihak pengembang.
- Supportability
Proses dapat didukung oleh teknologi semacam CASE tools.
- Acceptability
Penerimaan atas proses yang terdefinisi dan yang digunakan oleh software engineer selama pembangunan produk perangkat lunak.
- Reliability
Proses didisain dengan suatu metode untuk menghindari dari kesalahan, dan apabila ada kesalahan dapat terdeteksi sedini mungkin sebelum mengakibatkan cacat pada produk akhir.
- Robustness
Proses dapat terus dilanjutkan meskipun terdapat masalah yang tidak diharapkan muncul.
- Maintainability
Proses dapat mengadaptasi terhadap permintaan perubahan ataupun perbaikan
- Rapidity
Proses dapat diselesaikan dalam waktu yang relatif cepat

Model Proses RPL

Model proses disebut juga dengan aliran kerja (*workflow*), yakni tata cara bagaimana elemen-elemen proses berhubungan satu dengan lainnya. Aliran kerja ini dapat juga disebut dengan siklus hidup (*life-cycle*) sistem yang dimulai dari sejak sistem diajukan untuk dibangun hingga saat ia ditarik dari peredaran.

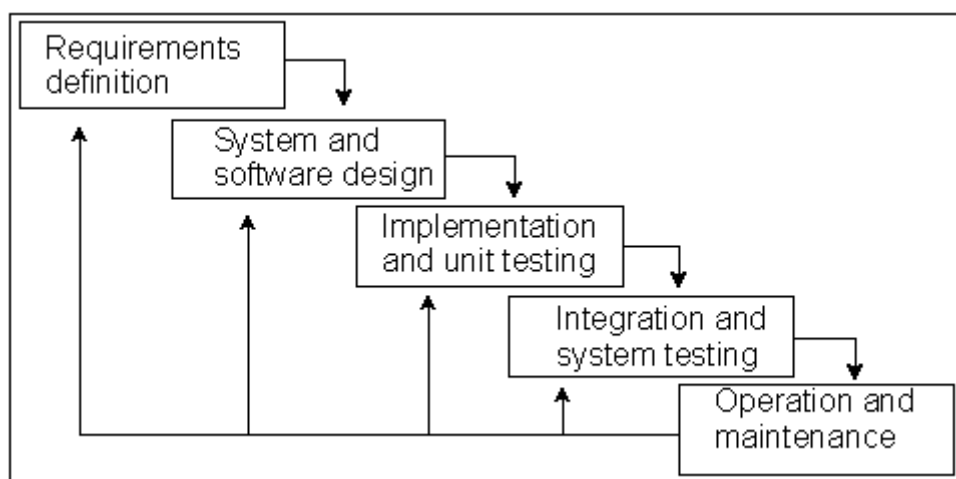
Fungsi utama model proses pengembangan perangkat lunak adalah :

- menentukan tahap-tahap yang diperlukan untuk pengembangan perangkat lunak.
- menentukan urutan pelaksanaan dari tahap-tahap tersebut dalam rangka pengembangan perangkat lunak.
- menentukan kriteria transisi/perpindahan dari satu tahap ke tahap berikutnya.

Untuk menentukan mana model yang terbaik, kita harus tahu apa kelebihan dan kekurangan model-model proses tersebut. Akan tetapi, model proses biasanya bukan ditentukan mana yang terbaik atau tidak, tetapi ditentukan oleh karakteristik dari berbagai macam faktor, misalnya tim SE-nya, atau software-nya sendiri, waktu untuk melakukan SE, kebijakan-kebijakan dari perusahaan, dan sebagainya. Dengan menggunakan model proses yang terbaru pun, ketika diaplikasikan ke dalam perusahaan misalnya, tetapi kalau tim SE-nya tidak siap dengan kondisi yang mengharuskan menggunakan model proses tersebut, tentunya tidak mungkin bisa dilakukan. Walaupun dipaksakan tentu saja hasilnya tidak akan maksimal. Akan tetapi di perusahaan lain, bisa jadi menerapkan model proses yang sama, tetapi hasilnya bagus, karena tim SE-nya siap atau scope softwrenya berbeda.

Model Waterfall

Nama model ini sebenarnya adalah “Linear Sequential Model”. Model ini sering disebut dengan “classic life cycle” atau model waterfall. Model ini adalah model yang muncul pertama kali yaitu sekitar tahun 1970 sehingga sering dianggap kuno, tetapi merupakan model yang paling banyak dipakai didalam Software Engineering (SE). Model ini melakukan pendekatan secara sistematis dan urut mulai dari level kebutuhan sistem lalu menuju ke tahap analisis, desain, coding, testing / verification, dan maintenance. Disebut dengan waterfall karena tahap demi tahap yang dilalui harus menunggu selesainya tahap sebelumnya dan berjalan berurutan. Sebagai contoh tahap desain harus menunggu selesainya tahap sebelumnya yaitu tahap requirement. Secara umum tahapan pada model waterfall (menurut Ian Sommerville) dapat dilihat pada gambar berikut :



Gambar di atas adalah tahapan umum dari model proses ini menurut Ian Sommerville. Penjelasan dari tiap tahapan tersebut adalah :

1. Requirements analysis and definition:

- Di tahapan ini dilakukan Analisa kebutuhan
2. System and software design:
Pada tahapan ini, desain dikerjakan setelah kebutuhan selesai dikumpulkan secara lengkap.
 3. Implementation and unit testing:
 4. Desain program diterjemahkan ke dalam kode-kode dengan menggunakan bahasa pemrograman yang sudah ditentukan. Program yang dibangun langsung diuji baik secara unit.
 5. Integration and system testing:
Penyatuan unit-unit program kemudian diuji secara keseluruhan (system testing).
 6. Operation and maintenance:
Mengoperasikan program dilingkungannya dan melakukan pemeliharaan, seperti penyesuaian atau perubahan karena adaptasi dengan situasi sebenarnya

Akan tetapi Roger S. Pressman memecah model ini menjadi 6 tahapan meskipun secara garis besar sama dengan tahapan-tahapan model waterfall pada umumnya. Berikut adalah penjelasan dari tahap-tahap yang dilakukan di dalam model ini menurut Pressman:

1. System / Information Engineering and Modeling. Pemodelan ini diawali dengan mencari kebutuhan dari keseluruhan sistem yang akan diaplikasikan ke dalam bentuk software. Hal ini sangat penting, mengingat software harus dapat berinteraksi dengan elemen-elemen yang lain seperti hardware, database, dsb. Tahap ini sering disebut dengan Project Definition.
2. Software Requirements Analysis. Proses pencarian kebutuhan diintensifkan dan difokuskan pada software. Untuk mengetahui sifat dari program yang akan dibuat, maka para software engineer harus mengerti tentang domain informasi dari software, misalnya fungsi yang dibutuhkan, user interface, dsb. Dari 2 aktivitas tersebut (pencarian kebutuhan sistem dan software) harus didokumentasikan dan ditunjukkan kepada pelanggan.
3. Design. Proses ini digunakan untuk mengubah kebutuhan-kebutuhan diatas menjadi representasi ke dalam bentuk “blueprint” software sebelum coding dimulai. Desain harus dapat mengimplementasikan kebutuhan yang telah disebutkan pada tahap sebelumnya. Seperti 2 aktivitas sebelumnya, maka proses ini juga harus didokumentasikan sebagai konfigurasi dari software.
4. Coding. Untuk dapat dimengerti oleh mesin, dalam hal ini adalah komputer, maka desain tadi harus diubah bentuknya menjadi bentuk yang dapat dimengerti oleh mesin, yaitu ke dalam bahasa pemrograman melalui proses coding. Tahap ini merupakan implementasi dari tahap design yang secara teknis nantinya dikerjakan oleh programmer.
5. Testing / Verification. Sesuatu yang dibuat haruslah diujicobakan. Demikian juga dengan software. Semua fungsi-fungsi software harus diujicobakan, agar software bebas dari error, dan hasilnya harus benar-benar sesuai dengan kebutuhan yang sudah didefinisikan sebelumnya.
6. Maintenance. Pemeliharaan suatu software diperlukan, termasuk di dalamnya adalah pengembangan, karena software yang dibuat tidak selamanya hanya seperti itu. Ketika dijalankan mungkin saja masih ada errors kecil yang tidak ditemukan sebelumnya, atau ada penambahan fitur-fitur yang belum ada pada software tersebut. Pengembangan diperlukan ketika adanya perubahan dari eksternal perusahaan seperti ketika ada pergantian sistem operasi, atau perangkat lainnya.

Kelebihan Waterfall :

- Model ini paling banyak digunakan karena pengaplikasian menggunakan model ini mudah

- ketika semua kebutuhan sistem dapat didefinisikan secara utuh, eksplisit, dan benar di awal project, maka SE dapat berjalan dengan baik dan tanpa masalah.

Meskipun seringkali kebutuhan sistem tidak dapat didefinisikan secara eksplisit yang diinginkan, tetapi paling tidak, problem pada kebutuhan sistem di awal project lebih ekonomis dalam hal uang (lebih murah), usaha, dan waktu yang terbuang lebih sedikit jika dibandingkan problem yang muncul pada tahap-tahap selanjutnya.

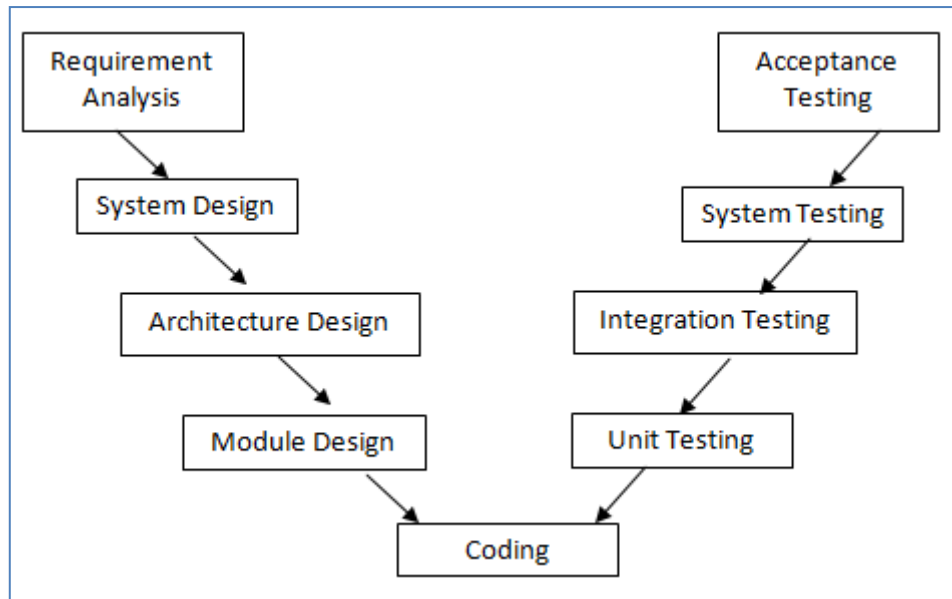
Kelemahan Waterfall :

- Ketika problem muncul, maka proses berhenti, karena tidak dapat menuju ke tahapan selanjutnya. Bahkan jika kemungkinan problem tersebut muncul akibat kesalahan dari tahapan sebelumnya, maka proses harus membenahi tahapan sebelumnya agar problem ini tidak muncul. Hal-hal seperti ini yang dapat membuang waktu pengerjaan SE.
- Karena pendekatannya secara sequential, maka setiap tahap harus menunggu hasil dari tahap sebelumnya. Hal itu tentu membuang waktu yang cukup lama, artinya bagian lain tidak dapat mengerjakan hal lain selain hanya menunggu hasil dari tahap sebelumnya. Oleh karena itu, seringkali model ini berlangsung lama pengerjaannya.
- Pada setiap tahap proses tentunya dikerjakan sesuai spesialisasinya masing-masing. Oleh karena itu, ketika tahap tersebut sudah tidak dikerjakan, maka sumber dayanya juga tidak terpakai lagi. Oleh karena itu, seringkali pada model proses ini dibutuhkan seseorang yang “multi-skilled”, sehingga minimal dapat membantu pengerjaan untuk tahapan berikutnya.

Berdasarkan kelebihan dan kekurangan tersebut, nantinya akan dikembangkan model-model yang lain, bahkan ada tahap evolusioner dari suatu model proses untuk mengatasi kelemahan-kelemahan tadi. Meskipun secara tahapan masih menggunakan standar tahapan waterfall model. Kesimpulannya adalah ketika suatu project skalanya sedang mengarah kecil bisa menggunakan model ini. Akan tetapi kalau sudah project besar, tampaknya kesulitan jika menggunakan model ini.

Model V

Teknik model V sering disebut sebagai pengembangan dari teknik waterfall. Disebut sebagai perluasan karena tahap-tahapnya mirip dengan yang terdapat dalam model waterfall. Jika dalam model waterfall proses dijalankan secara linear, maka dalam model V proses dilakukan bercabang. Dalam model V ini digambarkan hubungan antara tahap pengembangan software dengan tahap pengujiannya. “V” untuk verifikasi dan validasi dan merupakan model standar yang banyak dipakai di negara-negara Eropa seperti standar untuk proyek pertahanan dan administrasi federal di Jerman.



Berikut penjelasan masing-masing tahap beserta tahap pengujiannya:
(diambil www.softwaretestingvideos.com)

1. Requirement Analysis & Acceptance Testing
Tahap Requirement Analysis sama seperti yang terdapat dalam model waterfall. Keluaran dari tahap ini adalah dokumentasi kebutuhan pengguna. Acceptance Testing merupakan tahap yang akan mengkaji apakah dokumentasi yang dihasilkan tersebut dapat diterima oleh para pengguna atau tidak.
2. System Design & System Testing
Dalam tahap ini analis sistem mulai merancang sistem dengan mengacu pada dokumentasi kebutuhan pengguna yang sudah dibuat pada tahap sebelumnya. Keluaran dari tahap ini adalah spesifikasi software yang meliputi organisasi sistem secara umum, struktur data, dan yang lain. Selain itu tahap ini juga menghasilkan contoh tampilan window dan juga dokumentasi teknik yang lain seperti Entity Diagram dan Data Dictionary.
3. Architecture Design & Integration Testing
Sering juga disebut High Level Design. Dasar dari pemilihan arsitektur yang akan digunakan berdasar kepada beberapa hal seperti: pemakaian kembali tiap modul, ketergantungan tabel dalam basis data, hubungan antar interface, detail teknologi yang dipakai.
4. Module Design & Unit Testing
Sering juga disebut sebagai Low Level Design. Perancangan dipecah menjadi modul-modul yang lebih kecil. Setiap modul tersebut diberi penjelasan yang cukup untuk memudahkan programmer melakukan coding. Tahap ini menghasilkan spesifikasi program seperti: fungsi dan logika tiap modul, pesan kesalahan, proses input-output untuk tiap modul, dan lain-lain.
5. Coding
Dalam tahap ini dilakukan pemrograman terhadap setiap modul yang sudah dibentuk.

Kelebihan model V :

- Merupakan model pengembangan terstruktur.
- Setiap fase dapat diimplementasikan dengan dokumentasi yang detail dari fase sebelumnya.

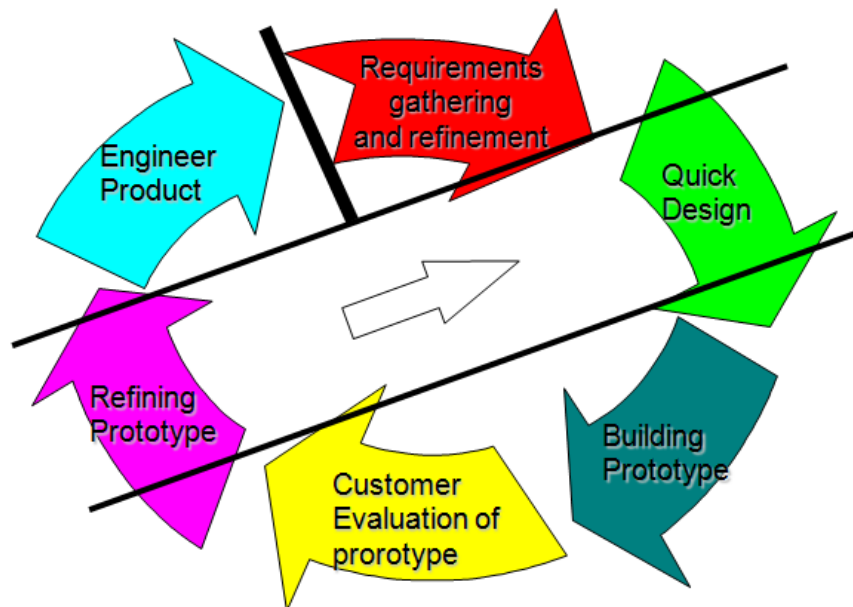
- Aktivitas pengujian dapat dimulai di awal proyek, sehingga mengurangi waktu proyek.

Kelemahan model V :

- Dokumentasi harus cukup detail agar fase selanjutnya dapat berjalan dengan baik.

Model Prototyping

Prototype adalah salah satu pendekatan dalam rekayasa perangkat lunak yang secara langsung mendemonstrasikan bagaimana sebuah perangkat lunak atau komponen-komponen perangkat lunak akan bekerja dalam lingkungannya sebelum tahapan konstruksi aktual dilakukan.

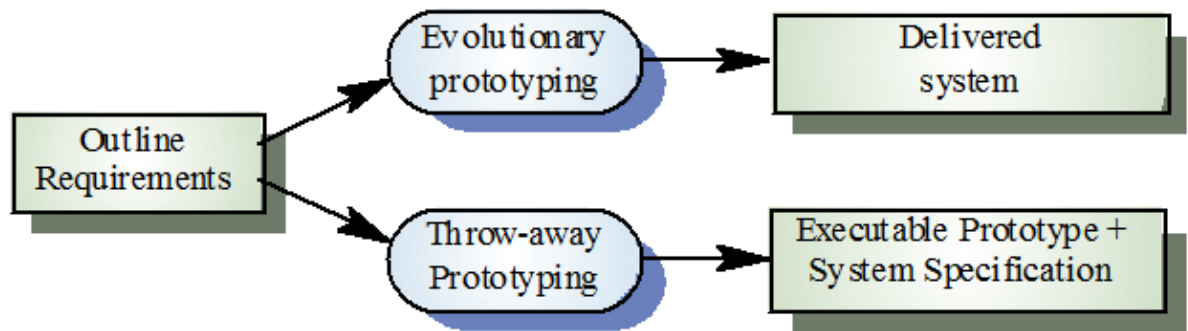


Prototyping tepat digunakan saat tidak didapati kepastian dimana definisi user masih sangat bersifat umum serta tidak rinci sehingga pengembang tidak tahu pasti mengenai :

- pilihan algoritma yang akan dipakai
- lingkungan sistem yang akan dipakai serta
- bentuk dan karakteristik antar muka pemakai.

Model prototype dimulai dengan pengumpulan kebutuhan. Pengembang dan pelanggan bertemu dan mendefinisikan obyektif keseluruhan dari perangkat lunak dan mengidentifikasi segala kebutuhan yang diketahui. Secara ideal prototipe berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan perangkat lunak. Prototipe bisa menjadi paradigma yang efektif bagi rekayasa perangkat lunak. Kuncinya adalah mendefinisikan aturan-aturan main pada saat awal, yaitu pelanggan dan pengembang keduanya harus setuju bahwa prototipe dibangun untuk berfungsi sebagai mekanisme pendefinisian kebutuhan.

Model prototipe :



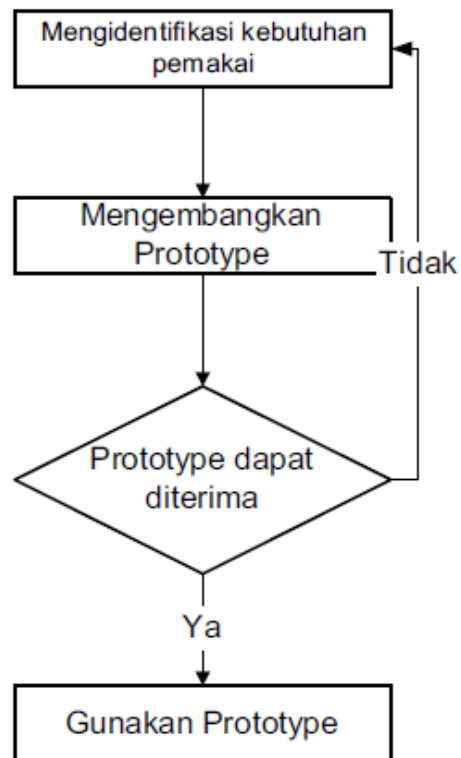
1. Evolutionary prototipe

Pada metode ini, prototype-nya tidak dibuang, melainkan digunakan sebagai dasar untuk iterasi perancangan selanjutnya. Dalam hal ini, sistem yang sesungguhnya dipandang sebagai evolusi dari versi awal yang terbatas menuju produk akhirnya. Tujuan dari evolutionary prototyping adalah menyerahkan sistem yang dapat dipakai kepada end-user. Pembangunan sistem di mulai dari kebutuhan yang paling dipahami. Prototipe evolusioner, sistem dibangun dengan cara memulai dari versi initial sampai versi final.

Tahapan model evolutionary prototipe :

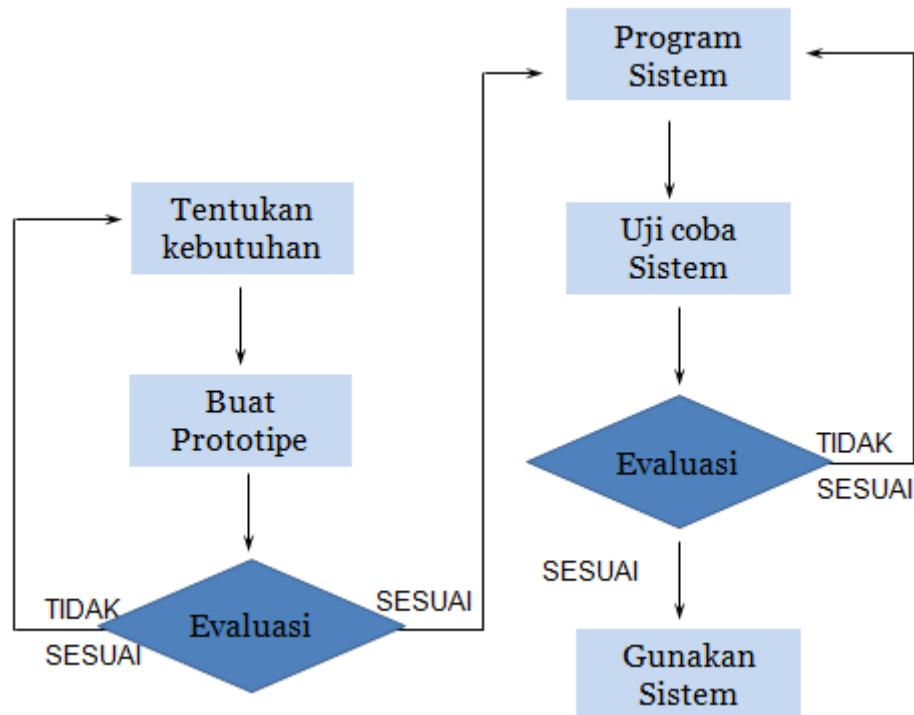
- a. Mengidentifikasi kebutuhan pemakai. Analis sistem mewawancarai pemakai untuk mendapatkan gagasan dari apa yang diinginkan pemakai terhadap sistem
- b. Mengembangkan prototipe. Analis sistem, mungkin bekerjasama dengan spesialis informasi lain, menggunakan satu atau lebih peralatan prototyping untuk mengembangkan sebuah prototipe.
- c. Menentukan apakah prototipe dapat diterima (sudah sesuai). Analis mendidik pemakai dalam penggunaan prototipe dan memberikan kesempatan kepada pemakai untuk membiasakan diri dengan sistem. Pemakai memberikan masukan bagi analis apakah prototipe memuaskan/sudah sesuai. Jika ya, langkah d. Akan diambil; jika tidak prototipe direvisi dengan mengulangi langkah a., b., c. Dengan pengertian yang lebih baik mengenai kebutuhan pemakai.
- d. Menggunakan prototipe. Prototipe ini menjadi sistem operasional.





2. Throw away prototipe

Prototype dibuat dan dites. Pengalaman yang diperoleh dari pembuatan prototype tersebut digunakan untuk membuat produk akhir (final), sementara prototype tersebut dibuang (tak dipakai). Tujuan throw-away prototyping adalah memvalidasi dan menurunkan persyaratan sistem. Prototipe dapat di mulai dari kebutuhan yang paling tidak dipahami. Prototipe Throw-away mencakup pengembangan prototype untuk memahami persyaratan sistem.



Kelebihan Prototyping :

- Kesalahpahaman antara sistem developer dan sistem user dapat diidentifikasi dan dibetulkan.
- Prototipe yang sedang bekerja mungkin sangat berguna dalam suatu pembuktian manajemen dimana suatu proyek adalah fleksibel sehingga menjamin kelangsungan dukungan.

Kelemahan Prototyping :

- Prototipe hanya bisa berhasil jika pemakai bersungguh-sungguh dalam menyediakan waktu dan pikiran untuk mengerjakan prototipe
- Kemungkinan dokumentasi terabaikan karena pengembangan lebih berkonsentrasi pada pengujian dan pembuatan prototipe
- Mengingat target waktu yang pendek, ada kemungkinan sistem yang dibuat tidak lengkap dan bahkan sistem kurang teruji
- Jika terlalu banyak proses pengulangan dalam membuat prototipe, ada kemungkinan pemakai menjadi jenuh dan memberikan reaksi yang negatif
- Apabila tidak terkelola dengan baik, maka prototipe menjadi tidak pernah berakhir. Hal ini disebabkan permintaan terhadap perubahan terlalu mudah untuk dipenuhi.

Model Spiral

Model ini ditemukan sekitar tahun 1988 oleh Barry Boehm. Spiral model adalah salah satu bentuk evolusi yang menggunakan metode iterasi natural yang dimiliki oleh model prototyping dan digabungkan dengan aspek sistematis yang dikembangkan dengan model waterfall.

Model ini juga mengkombinasikan top-down design dengan bottom-up design, dimana top-down design menetapkan sistem global terlebih dahulu, baru diteruskan dengan

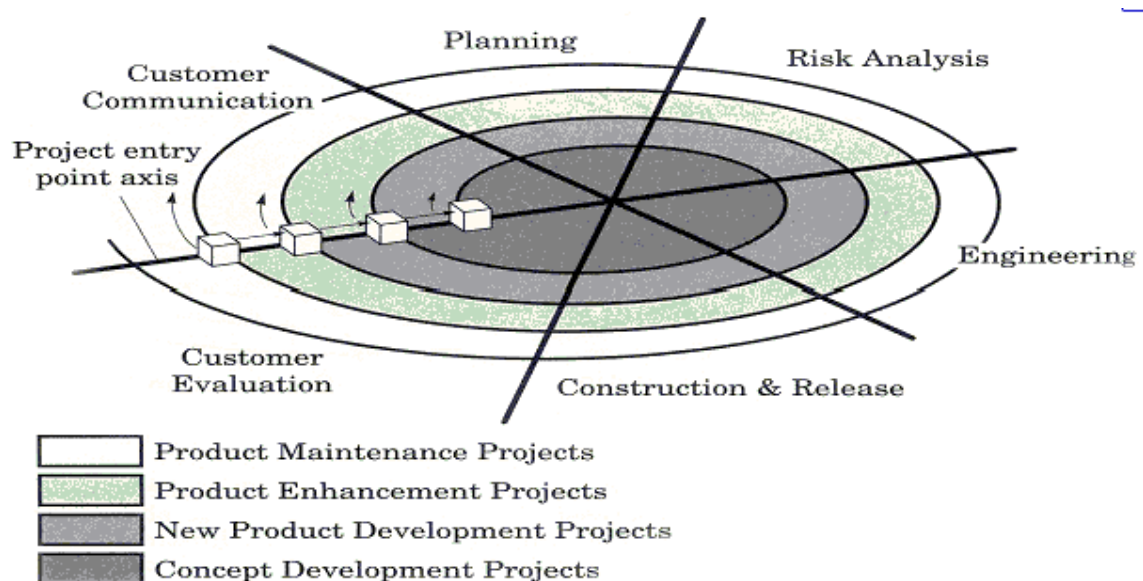
detail sistemnya, sedangkan bottom-up design berlaku sebaliknya. Top-down design biasanya diaplikasikan pada model waterfall dengan sequential-nya, sedangkan bottom-up design biasanya diaplikasikan pada model prototyping dengan feedback yang diperoleh.

Dari 2 kombinasi tersebut, yaitu kombinasi antara desain dan prototyping, serta top-down dan bottom-up, yang juga diaplikasikan pada model waterfall dan prototype, maka spiral model ini dapat dikatakan sebagai model proses hasil kombinasi dari kedua model tersebut. Oleh karena itu, model ini biasanya dipakai untuk pembuatan software dengan skala besar dan kompleks.

Spiral model dibagi menjadi beberapa framework aktivitas. Berikut adalah aktivitas-aktivitas yang dilakukan dalam spiral model:

1. Customer communication. Aktivitas yang dibutuhkan untuk membangun komunikasi yang efektif antara developer dengan user / customer terutama mengenai kebutuhan dari customer.
2. Planning. Aktivitas perencanaan ini dibutuhkan untuk menentukan sumberdaya, perkiraan waktu pengerjaan, dan informasi lainnya yang dibutuhkan untuk pengembangan software.
3. Risk Analysis. Aktivitas analisis resiko ini dijalankan untuk menganalisis baik resiko secara teknis maupun secara manajerial. Tahap inilah yang mungkin tidak ada pada model proses yang juga menggunakan metode iterasi, tetapi hanya dilakukan pada spiral model.
4. Engineering. Aktivitas yang dibutuhkan untuk membangun 1 atau lebih representasi dari aplikasi secara teknis.
5. Construction & Release. Aktivitas yang dibutuhkan untuk develop software, testing, instalasi dan penyediaan user / customer support seperti training penggunaan software serta dokumentasi seperti buku manual penggunaan software.
6. Customer evaluation. Aktivitas yang dibutuhkan untuk mendapatkan feedback dari user / customer berdasarkan evaluasi mereka selama representasi software pada tahap engineering maupun pada implementasi selama instalasi software pada tahap construction and release.

Berikut adalah gambar dari spiral model secara umum :



Tidak seperti model-model konvensional dimana setelah SE selesai, maka model tersebut juga dianggap selesai. Akan tetapi hal ini tidak berlaku untuk spiral model, dimana model ini dapat digunakan kembali sepanjang umur dari software tersebut. Pada umumnya, spiral model digunakan untuk beberapa project seperti Concept Development Project (proyek pengembangan konsep), New Product Development Project (proyek pengembangan produk baru), Product Enhancement Project (proyek peningkatan produk), dan Product Maintenance Project (proyek pemeliharaan proyek). Keempat project tersebut berjalan berurutan mengitari sirkuit dari spiral. Sebagai contoh setelah suatu konsep dikembangkan dengan melalui aktivitas-aktivitas dari spiral model, maka dilanjutkan dengan proyek selanjutnya yaitu pengembangan produk baru, peningkatan produk, sampai pemeliharaan proyek. Semuanya melalui sirkuit-sirkuit dari spiral model.

Kelebihan Spiral

- Pengguna dan developer bisa memahami dengan baik software yang dibangun karena progress dapat diamati dengan baik.
Model ini sangat baik digunakan untuk pengembangan sistem software dengan skala besar. Karena progress perkembangan dari SE dapat dipantau oleh kedua belah pihak baik developer maupun user / customer, sehingga mereka dapat mengerti dengan baik mengenai software ini begitu juga dengan resiko yang mungkin didapat pada setiap aktivitas yang dilakukan.
- Estimasi menjadi lebih realistis seiring berjalannya proyek karena masalah ditemukan sesegera mungkin.
Kelebihan model ini ada pada analisis resiko yang dilakukan, sehingga resiko tersebut dapat direduksi sebelum menjadi suatu masalah besar yang dapat menghambat SE.
- Lebih mampu menangani perubahan yang sering terjadi pada software development.
Dengan menggunakan prototype juga bisa menghindari terjadinya resiko yang muncul, tetapi kelebihan dari model ini yaitu dilakukannya proses prototyping untuk setiap tahap dari evolusi produk secara kontinu. Model ini melakukan tahap2 yang sudah sangat baik didefinisikan pada model waterfall dan ditambah dengan iterasi yang menyebabkan model ini lebih realistis untuk merefleksikan dunia nyata.
- Software engineers bisa bekerja lebih cepat pada proyek.

Kelemahan Spiral

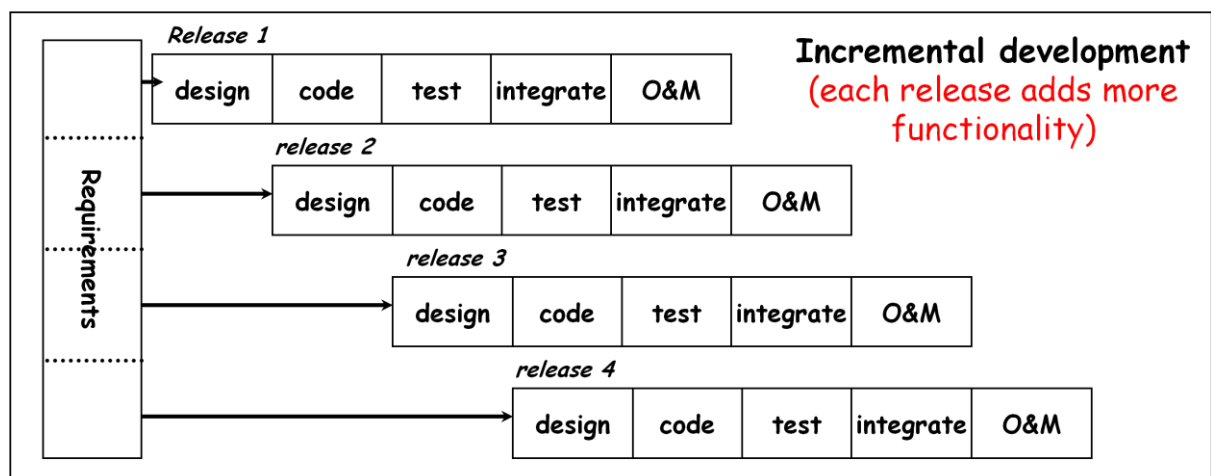
- Membutuhkan waktu yang lama dan dana yang besar.
- Membutuhkan planning jangka panjang yang baik agar program bisa selesai dengan baik.

Kekurangannya ada pada masalah pemikiran user / customer dimana mereka pada umumnya tidak yakin bahwa pendekatan evolusioner ini dapat terus dalam ambang kontrol yang bagus. Dibutuhkan kombinasi kemampuan manajerial dan teknis tersendiri untuk mengontrol model ini sehingga dengan sendirinya dapat meyakinkan user / customer tersebut. Mengenai analisis resiko yang terdapat pada model ini dibutuhkan kemampuan expert tersendiri agar tahap ini dapat berjalan dengan baik. Dibutuhkan kemampuan manajemen yang tinggi untuk melakukan perkiraan resiko, karena jika ada resiko yang luput untuk dievaluasi, dikhawatirkan dapat muncul di kemudian hari yang dapat menghambat proses SE.

Model Incremental

Model proses ini hampir sama dengan model proses Waterfall. Bedanya adalah model proses ini dilakukan secara bertahap dan tahap pengerjaan dilakukan permodul. Bisa dikatakan Incremental adalah bentuk pengulangan secara bertahap dari Waterfall. Model ini mengaplikasikan urutan-urutan linier secara bertingkat selaras dengan berjalannya waktu. Setiap urutan linier menghasilkan penambahan (increment) pada software yang dikirimkan. Proses model ini menfokuskan pada pengiriman produk operasional pada setiap penambahannya. Produk awal adalah versi rendah dari produk akhir namun telah mampu mengakomodir kebutuhan pengguna.

Apabila aliran proses dari Communication sampai Deployment telah selesai pada tahap pertama, maka dilanjutkan pada pengerjaan tahap kedua yang aliran prosesnya sama yaitu dari Communication sampai Deployment. Proses ini berlangsung berulang-ulang secara bertahap sampai tahap final. Tahap pertama adalah tahap yang penting karena tahap pertama merupakan tahap kunci dan produk perdana dalam pengembangan karena apabila produk pada tahap pertama gagal, maka tahap selanjutnya tidak akan berjalan. Tahap pertama sering disebut dengan Core Product.



Kelebihan Incremental

- penambahan kemampuan fungsional akan lebih mudah diuji, diverifikasi, dan divalidasi dan dapat menurunkan biaya yang dikeluarkan untuk memperbaiki sistem.

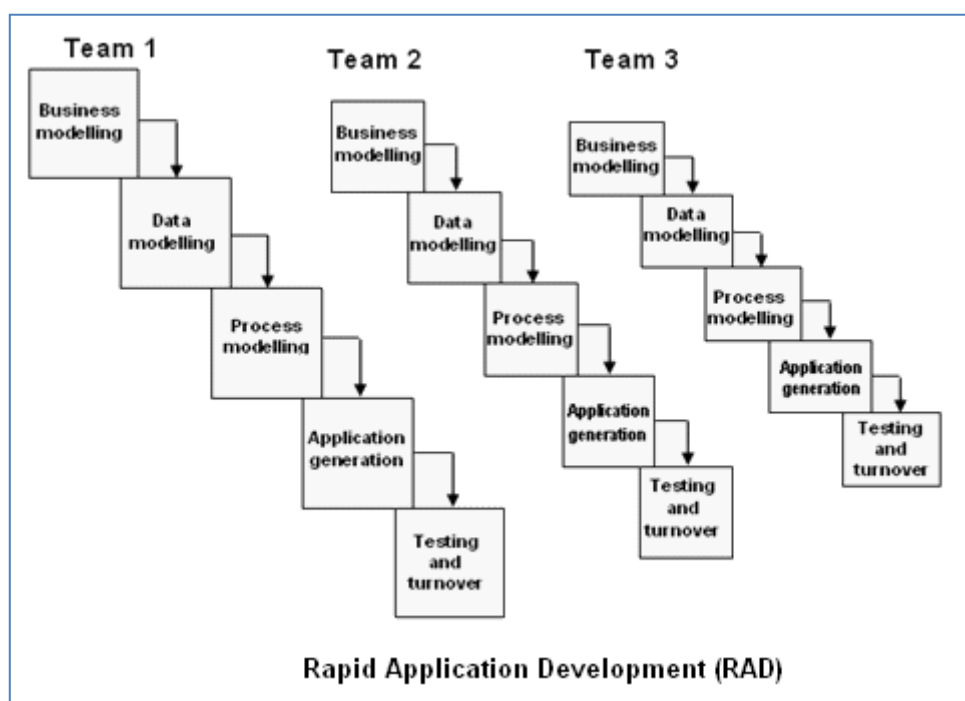
Kelemahan Incremental

- Hanya akan berhasil jika tidak ada staffing untuk penerapan secara menyeluruh
- Penambahan staf dilakukan jika hasil incremental akan dikembangkan lebih lanjut

Model RAD (Rapid Application Development)

Model proses ini merupakan pembangan dari model Incremental. Sama seperti Incremental tetapi waktu pengerjaan antara siklus pengembangannya berjalan secara singkat. Model RAD merupakan adaptasi *High-speed* dari model waterfall yang pengembangannya dilakukan dengan menggunakan pendekatan component-based. Maksud dari component-based pada RAD ini adalah agar proses pengerjaannya dapat dipercepat dengan cara membagi program menjadi bagian-bagian terkecil yang tiap bagian tersebut ada tim tersendiri yang bertanggung jawab.

Untuk proses skala besar, RAD Model membutuhkan SDM yang memadai untuk membentuk tim-tim yang diperlukan agar tahap pengerjaan berjalan dengan cepat. RAD Model ini tidak cocok dipakai pada pengerjaan proyek yang memiliki tingkat resiko teknikal yang tinggi. Hal ini bisa terjadi pada saat aplikasi baru menggunakan teknologi baru atau pada saat software yang baru memerlukan derajat kebergantungan yang tinggi terhadap program komputer yang sudah ada. RAD juga memerlukan pengembang dan pelanggan yang komitmen terhadap aktifitas yang ketat sesuai dengan time frame yang diberikan.



Model RAD menekankan pada fase-fase berikut :

1. Business modeling. Pada tahap ini, aliran informasi (information flow) pada fungsi-fungsi bisnis dimodelkan untuk mengetahui informasi apa yang mengendalikan proses bisnis, informasi apa yang dihasilkan, siapa yang membuat informasi itu, kemana saja informasi mengalir, dan siapa yang mengolahnya.
2. Data modeling. Aliran informasi yang didefinisikan dari business modeling, disaring lagi agar bisa dijadikan bagian-bagian dari objek data yang dibutuhkan untuk mendukung bisnis tersebut. Karakteristik (atribut) setiap objek ditentukan beserta relasi antar objeknya.
3. Process modeling. Objek-objek data yang didefinisikan sebelumnya diubah agar bisa menghasilkan aliran informasi untuk diimplementasikan menjadi fungsi bisnis. Pengolahan deskripsi dibuat untuk menambah, merubah, menghapus, atau mengambil kembali objek data.

4. Application generation. RAD bekerja dengan menggunakan fourth generation techniques (4GT). Sehingga pada tahap ini sangat jarang digunakan pemrograman konvensional menggunakan bahasa pemrograman generasi ketiga (third generation programming languages), tetapi lebih ditekankan pada reuse komponen-komponen (jika ada) atau membuat komponen baru (jika perlu). Dalam semua kasus, alat bantu untuk otomatisasi digunakan untuk memfasilitasi pembuatan perangkat lunak.
5. Testing and turnover. Karena menekankan pada penggunaan kembali komponen yang telah ada (reuse), sebagian komponen-komponen tersebut sudah diuji sebelumnya. Sehingga mengurangi waktu testing secara keseluruhan. Kecuali untuk komponen-komponen baru.

Kelebihan RAD :

RAD memang lebih cepat dari Waterfall. jika kebutuhan dan batasan project sudah diketahui dengan baik. Juga jika proyek memungkinkan untuk dimodularisasi.

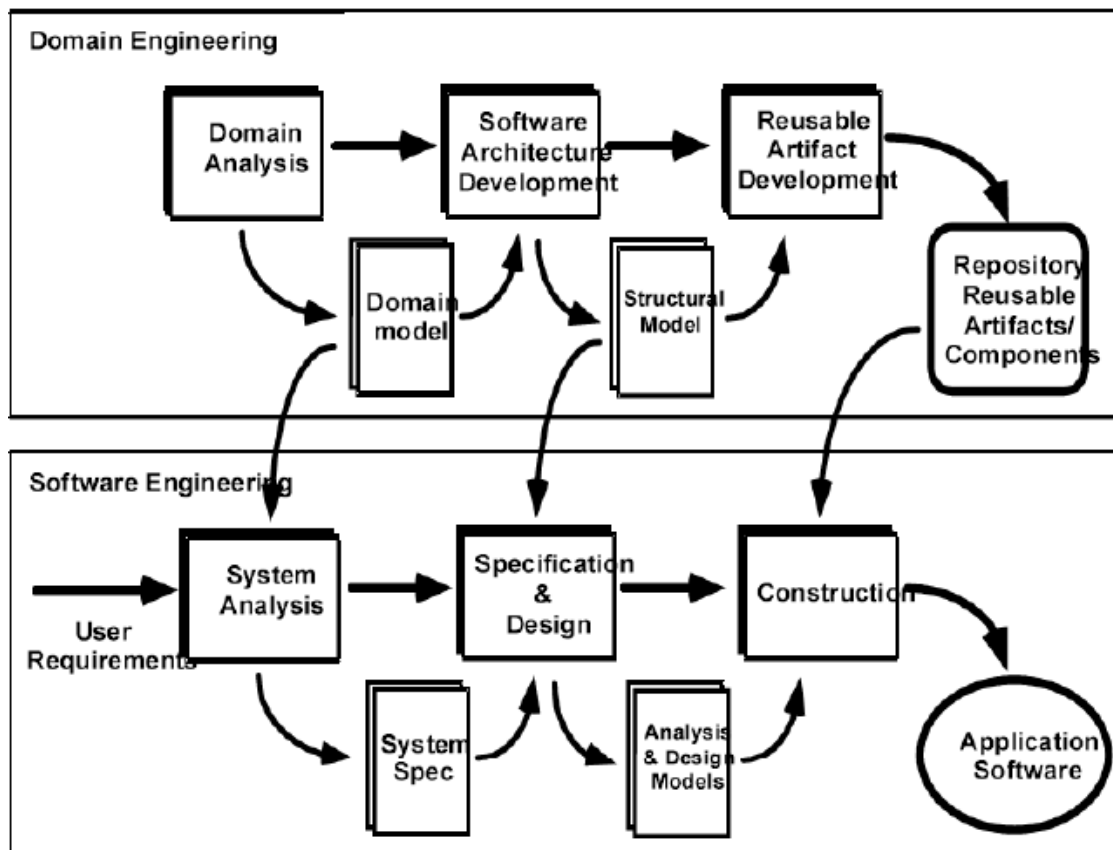
Kekurangan RAD :

- Tidak semua project bisa dipecah (dimodularisasi), sehingga belum tentu RAD dipakai pada semua proyek.
- Karena project dipecah menjadi beberapa bagian, maka dibutuhkan banyak orang untuk membentuk suatu tim yang mengerjakan tiap bagian tersebut.
- Membutuhkan komitmen antara pengembang dengan pelanggan.
- Karena dibuat dengan reuse komponen-komponen yang sudah ada, fasilitas-fasilitas pada tiap komponen belum tentu digunakan seluruhnya oleh program yang me-reuse-nya sehingga kualitas program bisa menurun.

Model CBSE (Component Based Software Engineering)

Model CBSE adalah proses yang menekankan perancangan dan pembangunan software dengan menggunakan komponen software yang sudah ada. Model ini bersifat iteratif atau berulang-ulang prosesnya. Tahapan CBSE terdiri dari dua bagian yang terjadi secara paralel yaitu :





1. Software engineering (component-based development)

Component based development sangat berkaitan dengan teknologi berorientasi objek. Pada pemrograman berorientasi objek, banyak class yang dibangun dan menjadi komponen dalam suatu software. Class-class tersebut bersifat reusable artinya bisa digunakan kembali. Penggunaan kembali komponen software yang sudah ada menguntungkan dari segi :

- Siklus waktu pengembangan software, karena mampu mengurangi waktu 70%
- Biaya produksi berkurang sampai 84% karena pembangunan komponen berkurang

Hal-hal yang dilakukan pada tahap software engineering (component-based development) adalah :

- Melakukan analisis terhadap domain model yang sudah ditetapkan
- Menentukan spesifikasi dan merancang berdasarkan model struktur dan spesifikasi sistem.
- Melakukan pembangunan software dengan menggunakan komponen-komponen yang sudah ditetapkan berdasarkan analisis dan rancangan yang dihasilkan sebelumnya hingga akhirnya menghasilkan software.

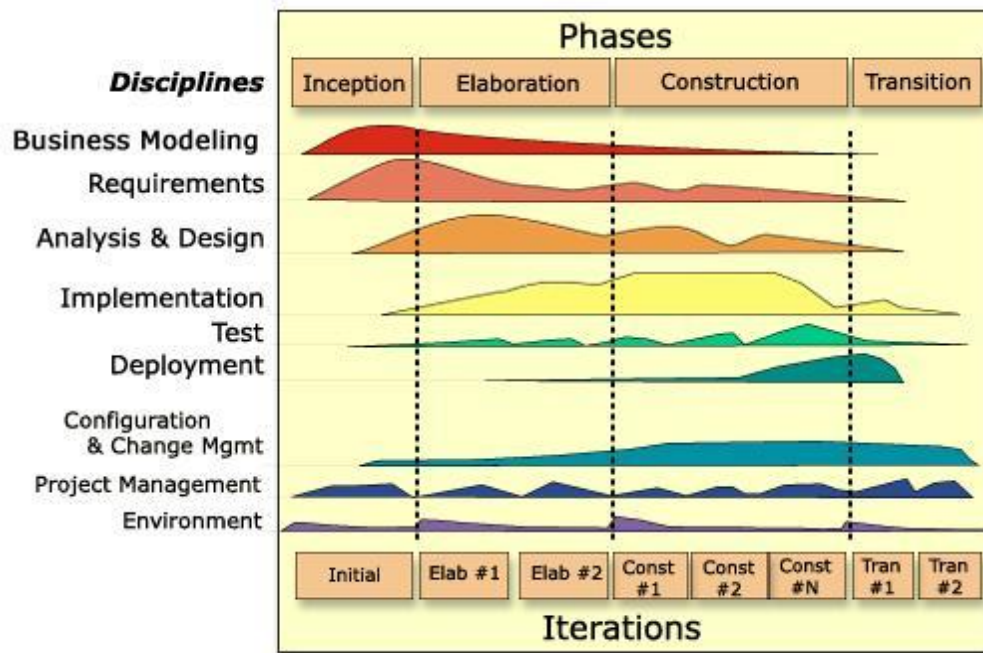
2. Domain engineering

Hal-hal yang dilakukan pada tahapan domain engineering:

- Menciptakan model domain bagi aplikasi yang akan digunakan untuk menganalisis kebutuhan pengguna.
- Identifikasi, pembangunan, pengelompokan dan pengalokasikan komponen-komponen software supaya bisa digunakan pada sistem yang ada dan yang akan datang.

Model Rational Unified Process (RUP)

Rational Unified Process (RUP) merupakan suatu pendekatan disiplin dalam mengerjakan tugas dan tanggung jawab melalui berbagai best practise di dalam suatu organisasi pengembangan pengembangan perangkat lunak. Tujuan dari RUP adalah untuk memastikan dihasilkan produk perangkat lunak dengan kualitas yang tinggi (minim error dan berjalan sesuai yang diharapkan), serta memenuhi semua kebutuhan stakeholder dengan biaya dan waktu yang sudah diprediksikan. menunjukkan secara keseluruhan kerangka kerja (framework) yang dimiliki RUP



RUP menggunakan konsep object oriented dengan aktifitas yang berfokus pada pengembangan model dengan menggunakan unified model language (UML). UML adalah bahasa standar untuk penulisan blueprint perangkat lunak. Model ini membagi suatu sistem aplikasi menjadi beberapa komponen sistem dan memungkinkan para developer aplikasi untuk menerapkan metoda iterative (analisis, disain, implementasi dan pengujian) pada tiap komponen. dapat dilihat bahwa RUP memiliki dua dimensi, yaitu:

1. Dimensi horisontal mewakili aspek-aspek dinamis dari pengembangan perangkat lunak. Aspek ini dijabarkan dalam tahapan pengembangan atau fase. Setiap fase akan memiliki suatu major milestone yang menandakan akhir fase dan awal dari fase selanjutnya. Setiap fase dapat terdiri dari satu atau beberapa iterasi.
2. Dimensi vertikal mewakili aspek-aspek statis dari proses pengembangan perangkat lunak yang dikelompokkan ke dalam beberapa disiplin. Proses pengembangan perangkat lunak yang dijelaskan kedalam beberapa disiplin terdiri dari empat elemen penting, yakni who is doing what, how and when.

Tahapan pengembangan perangkat lunak dalam RUP terbagi menjadi 4 fase, sebagai berikut :

- **Inception**
merupakan tahap untuk mengidentifikasi sistem yang akan dikembangkan. Aktivitas yang dilakukan pada tahap ini antara lain mencakup analisis sistem

eksisting, perumusan sistem target, penentuan arsitektur global target, identifikasi kebutuhan, perumusan persyaratan (fungsional, performansi, keamanan, GUI, dll.), perumusan kebutuhan pengujian (level unit, integrasi, sistem, performansi, fungsionalitas, keamanan, dll.), pemodelan diagram UML (diagram use case dan activity), dan pembuatan dokumentasi.

- **Elaboration**
merupakan tahap untuk melakukan disain secara lengkap berdasarkan hasil analisis di tahap inception. Aktivitas yang dilakukan pada tahap ini antara lain mencakup pembuatan disain arsitektur subsistem (architecture pattern), disain komponen sistem, disain format data (protokol komunikasi), disain database, disain antarmuka/tampilan, disain peta aliran tampilan, penentuan design pattern yang digunakan, pemodelan diagram UML (diagram sequence, class, component, deployment, dll.), dan pembuatan dokumentasi.
- **Construction**
merupakan tahap untuk mengimplementasikan hasil disain dan melakukan pengujian hasil implementasi. Pada tahap awal construction, ada baiknya dilakukan pemeriksaan ulang hasil analisis dan disain, terutama disain pada domain perilaku (diagram sequence) dan domain struktural (diagram class, component, deployment). Apabila disain yang dibuat telah sesuai dengan analisis sistem, maka implementasi dengan bahasa pemrograman tertentu dapat dilakukan. Aktivitas yang dilakukan pada tahap ini antara lain mencakup pengujian hasil analisis dan disain (misal menggunakan Class Responsibility Collaborator untuk kasus pemrograman berorientasi obyek), pendataan kebutuhan implementasi lengkap (berpedoman pada identifikasi kebutuhan di tahap analisis), penentuan coding pattern yang digunakan, pembuatan program, pengujian, optimasi program, pendataan berbagai kemungkinan pengembangan / perbaikan lebih lanjut, dan pembuatan dokumentasi.
- **Transition**
merupakan tahap untuk menyerahkan sistem aplikasi ke konsumen (roll-out), yang umumnya mencakup pelaksanaan pelatihan kepada pengguna dan testing beta aplikasi terhadap ekspektasi pengguna.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Tujuan perencanaan proyek.
Ruang lingkup proyek perangkat lunak. Sumber daya yang dibutuhkan. Estimasi proyek perangkat lunak.

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

Kode MK
87011

Disusun Oleh
Tim Dosen

04

Abstract

Membahas mengenai tujuan, ruang lingkup dan sumberdaya yang dibutuhkan dalam proyek perangkat lunak.

Kompetensi

Mahasiswa dapat membuat perencanaan proyek perangkat lunak.

Tujuan Perencanaan Proyek

Proses manajemen proyek perangkat lunak dimulai dengan beberapa aktivitas yang secara kolektif disebut dengan project planning (perencanaan proyek). Aktivitas ini dimulai dengan estimasi, yang merupakan gambaran dimana kita melihat masa depan serta menerima tingkat ketidakpastian sebagai bahan pembicaraan. Perencanaan proyek memberikan sebuah peta jalan bagi suksesnya rekayasa perangkat lunak.

Observasi pada Estimasi

Estimasi yang diperlukan dalam perancangan proyek perangkat lunak di antaranya adalah sumber daya, biaya, dan jadwal sebagai usaha dalam pengembangan perangkat lunak, mengakses informasi historis yang baik, dan keberanian untuk melakukan pengukuran kuantitatif bila hanya data kualitatif saja yang ada. Berikut adalah yang menimbulkan ketidakpastian dalam estimasi :

- Project complexity (kompleksitas proyek) berpengaruh kuat terhadap ketidakpastian yang inheren dalam perencanaan. Kompleksitas ini merupakan pengukuran relatif yang dipengaruhi oleh kebiasaan dengan usaha yang dilakukan sebelumnya.
- Project size (Ukuran proyek) Merupakan faktor penting yang dapat mempengaruhi akurasi estimasi. Bila ukuran bertambah maka ketergantungan di antara berbagai elemen perangkat lunak akan meningkat dengan cepat.
- Structural uncertainty (Ketidakpastian struktural) Tingkat ketidakpastian struktural juga berpengaruh dalam risiko estimasi. Dengan melihat kembali, kita dapat mengingat lagi hal-hal yang terjadi dan dapat menghindari tempat-tempat dimana masalah muncul.

Risiko diukur melalui tingkat ketidakpastian pada estimasi kuantitatif yang dibuat untuk sumber daya, biaya, dan jadwal. Bila ruang lingkup proyek atau syarat proyek tidak dipahami dengan baik, maka risiko dan ketidakpastian menjadi sangat tinggi. Perencana perangkat lunak harus melengkapi fungsi, kinerja, dan definisi interface (yang diisikan ke dalam spesifikasi sistem). Pendekatan-pendekatan rekayasa perangkat lunak modern (seperti model proses evolusioner) memakai pandangan pengembangan yang interaktif. Dengan pandangan semacam ini dimungkinkan untuk melihat estimasi dan merevisinya bila customer mengubah kebutuhannya.

Tujuan Perencanaan Proyek

Tujuan perencanaan proyek perangkat lunak adalah untuk menyediakan sebuah kerangka kerja yang memungkinkan manajer membuat estimasi yang dapat dipertanggungjawabkan mengenai sumber daya, biaya dan jadwal. Tujuan perencanaan dicapai melalui suatu proses penemuan informasi yang menunjuk ke estimasi yang dapat dipertanggungjawabkan.

Ruang Lingkup Perangkat Lunak

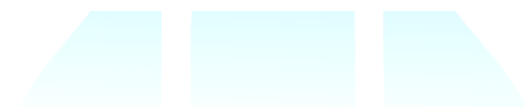
Penentuan ruang lingkup perangkat lunak merupakan aktivitas pertama dalam perencanaan proyek perangkat lunak. Ruang lingkup perangkat lunak menggabungkan fungsi, kinerja, batasan, interface dan reliabilitas. Fungsi yang digambarkan dalam statmen ruang lingkup dievaluasi dan disaring untuk memberikan awalan yang lebih detail pada saat estimasi dimulai. Pertimbangan kinerja melingkupi pemrosesan dan kebutuhan waktu respon. Batasan ini mengidentifikasi dari batas yang ditempatkan pada perangkat lunak oleh perangkat keras eksternal, memori, atau sistem informasi yang ada.

Mencari Informasi Yang Dibutuhkan Untuk Ruang Lingkup

Teknik yang banyak dipakai secara umum untuk menjembatani jurang komunikasi antara pelanggan dan pengembang serta untuk memulai proses komunikasi adalah dengan melakukan pertemuan atau wawancara pendahuluan. Gause & Weinberg mengusulkan bahwa analisis harus dimulai dengan mengajukan pertanyaan-pertanyaan bebas konteks, yaitu serangkaian pertanyaan yang akan membawa pada pemahaman mendasar terhadap masalah, orang yang menginginkan suatu solusi, sifat solusi yang diharapkan, dan efektivitas pertemuan itu. Beberapa pertanyaan bebas konteks pada pelanggan yang meliputi tujuan keseluruhan, serta keuntungan :

- o Siapa di belakang permintaan kerja ini?
- o Siapa yang akan memakai solusi ini?
- o Apakah yang akan menjadi keuntungan ekonomi dari sebuah solusi yang sukses?
- o Adakah sumberdaya lain bagi solusi ini?

Beberapa contoh pertanyaan yang memungkinkan analisis untuk memahami masalah lebih baik :



- o Bagaimanakah anda menandai output yang baik yang akan dimunculkan oleh sebuah solusi yang baik?
- o Masalah apa yang akan dituju oleh solusi ini?
- o Dapatkah anda memperlihatkan atau menggambarkan lingkungan di mana solusi akan dipakai?
- o Adakah batasan atau isu kinerja khusus yang akan mempengaruhi cara pendekatan terhadap solusi?

Beberapa pertanyaan yang berfokus pada efektivitas pertemuan :

- o Apakah anda orang yang tepat untuk menjawab pertanyaan ini? Apakah anda resmi?
- o Apakah pertanyaan saya relevan dengan problem yang anda punyai?
- o Apakah saya terlalu banyak pertanyaan?
- o Apakah ada orang lain yang dapat menyediakan informasi tambahan?
- o Adakah sesuatu yang lain yang dapat saya tanyakan kepada anda?

Bagian Question dan Answer hanya akan digunakan untuk pertemuan pertama yang kemudian diganti dengan format pertemuan yang mengkombinasikan elemen-elemen penyelesaian masalah, negosiasi, dan spesifikasi. Sejumlah peneliti lepas mengembangkan pendekatan yang berorientasi pada tim terhadap pengumpulan kebutuhan yang dapat diterapkan untuk membangun ruang lingkup sebuah proyek, yang disebut teknik spesifikasi aplikasi yang terapan (FAST)

Sumber Daya

Mengestimasi sumber daya yang dibutuhkan untuk menyelesaikan usaha pengembangan perangkat lunak yang meliputi manusia, komponen perangkat lunak, dan peranti perangkat keras/perangkat lunak.

memperlihatkan sumber daya pengembangan sebagai sebuah piramid. Peranti perangkat keras dan perangkat lunak berada pada fondasi dari piramida dan menyediakan infrastruktur untuk mendukung usaha pengembangan (lingkungan pengembang).

Dalam tingkat yang lebih tinggi terdapat komponen perangkat lunak reusable – blok bangunan perangkat lunak yang dapat mengurangi biaya pengembangan secara dramatis dan mempercepat penyampaian. Dan di puncak terdapat sumber daya utama yaitu manusia. Masing-masing sumber daya ditentukan dengan empat karakteristik :

- o Deskripsi sumber daya

- o Statemen ketersediaan
- o Waktu kronologis sumber daya diperlukan
- o Durasi waktu sumber daya diaplikasikan

Sumber daya manusia

Perencanaan sumber daya manusia dimulai dengan mengevaluasi ruang lingkup serta memilih kecakapan yang dibutuhkan untuk menyelesaikan pengembangan. Baik posisi organisasi maupun specialty. Jumlah orang yang diperlukan untuk sebuah proyek perangkat lunak dapat ditentukan setelah estimasi usaha pengembangan dibuat.

Sumber daya perangkat lunak reusable

Kreasi dan penggunaan kembali blok bangunan perangkat lunak yang seharusnya dikatalog menjadi referensi yang mudah, distandarisasi untuk aplikasi yang mudah, dan divalidasi untuk integrasi yang mudah. Ada empat kategori sumber daya perangkat lunak yang harus dipertimbangkan pada saat perencanaan berlangsung, yaitu :

- Komponen off-the-self Perangkat lunak yang ada dapat diperoleh dari bagian ketiga atau telah dikembangkan secara internal untuk proyek sebelumnya.
- Komponen full-experience Spesifikasi, kode, desain atau pengujian data yang sudah ada yang dikembangkan pada proyek yang lalu yang serupa dengan perangkat lunak yang akan dibangun pada proyek saat ini.
- Komponen partial-experience Aplikasi, kode, desain, atau data pengujian yang ada pada proyek yang lalu yang dihubungkan dengan perangkat lunak yang dibangun untuk proyek saat ini, tetapi akan membutuhkan modifikasi substansial.
- Komponen baru Komponen perangkat lunak yang harus dibangun oleh tim perangkat lunak khususnya adalah untuk kebutuhan proyek sekarang .

Lebih baik mengkhususkan syarat sumber daya perangkat lunak dari awal. Dengan cara ini evaluasi teknis dari semua alternatif dapat dilakukan dan akuisisi secara berkala dapat terjadi.

Sumber daya lingkungan

Lingkungan yang mendukung proyek perangkat lunak, yang disebut juga software engineering environment (SEE), menggabungkan perangkat lunak dan perangkat keras. Karena sebagian besar organisasi perangkat lunak memiliki konstituen ganda yang memerlukan akses ke SEE, maka perencana proyek harus menentukan jendela waktu yang dibutuhkan bagi perangkat keras dan perangkat lunak serta membuktikan bahwa sumber-sumber daya tersebut dapat diperoleh.

Pada saat sebuah sistem berbasis komputer akan direkayasa, tim perangkat lunak mungkin membutuhkan akses ke elemen perangkat keras yang sedang dikembangkan oleh tim rekayasa yang lain.

Estimasi Proyek Perangkat Lunak

Biaya perangkat lunak terdiri dari presentase kecil pada biaya sistem berbasis komputer secara keseluruhan. Kesalahan estimasi biaya yang besar dapat memberikan perbedaan antara keuntungan dan kerugian. Estimasi proyek perangkat lunak dapat ditransformasi dari suatu seni yang misterius ke dalam langkah-langkah yang sistematis yang memberikan estimasi dengan risiko yang dapat diterima. Sejumlah pilihan untuk mencapai estimasi biaya dan usaha yang dapat dipertanggung jawabkan :

1. Menunda estimasi sampai akhir proyek
2. Mendasarkan estimasi pada proyek-proyek yang mirip yang sudah pernah dilakukan sebelumnya
3. Menggunakan “teknik dekomposisi” yang relatif sederhana untuk melakukan estimasi biaya dan usaha proyek
4. Menggunakan satu atau lebih model empiris bagi estimasi usaha dan biaya perangkat lunak.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Analisis kebutuhan perangkat lunak. Teknik komunikasi dan prinsip analisis. Pembuatan model prototype perangkat lunak.

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka
05

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Membahas mengenai Analisa kebutuhan yang dilakukan agar perangkat lunak yang dibuat dapat memenuhi.

Kompetensi

Mahasiswa dapat menganalisa, menguasai teknik perangkat lunak dan membuatnya.

Analisis Kebutuhan Perangkat Lunak

Definisi

Proses manajemen proyek perangkat lunak dimulai dengan kegiatan project planning (perencanaan proyek). Yang pertama dari aktifitas ini adalah estimation (perkiraan). Estimasi membawa resiko yang inheren (dari diri sendiri) dan resiko inilah yang membawa ketidakpastian. Yang mempengaruhi estimasi :

- Project complexity (kompleksitas proyek)
- Project size (ukuran proyek)
- Struktural uncertainty (ketidakpastian struktural)

Tujuan Perencanaan Proyek Perangkat Lunak :

menyediakan sebuah kerangka kerja yang memungkinkan manajer membuat estimasi yang dapat dipertanggungjawabkan terhadap sumber daya, biaya dan jadwal pada awal proyek yang dibatasi oleh waktu.

Aktifitas Perencanaan Proyek PL

1. Menentukan ruang lingkup PL
2. Mengestimasi sumber daya yang dibutuhkan

Ruang Lingkup PL

Ruang lingkup PL menggambarkan : fungsi, kinerja, batasan, interface dan reliabilitas. Fungsi yang digambarkan dlm statemen ruang lingkup dievaluasi untuk memberikan awalan yang lebih detail pada saat dimulai estimasi.

Kinerja melingkupi pemrosesan dan kebutuhan waktu respon. Batasan mengidentifikasi batas yang ditempatkan pada PL oleh perangkat keras eksternal, memori atau sistem lain.

Informasi yang dibutuhkan (awal pertemuan antara pelanggan dan pengembang) *

Pertanyaan berfokus pada pelanggan, tujuan keseluruhan serta keuntungan.

- Siapa di belakang permintaan kerja ini?
- Siapa yang akan memakai solusi ini?
- Apakah keuntungan ekonomi dari solusi yang sukses?
- Adakah sumber daya lain bagi solusi ini? * Pertanyaan yang memungkinkan analis memahami masalah lebih baik dan pelanggan menyuarakan persepsi tentang sebuah solusi.
- Bagaimana Anda (pelanggan) menandai output yg baik yg akan dihasilkan oleh sebuah solusi yg baik?
- Masalah apa yang dituju solusi ini?
- Dapatkah anda menggambarkan lingkungan dimana solusi akan dipakai?
- Adakah batasan atau isu kinerja khusus yg akan mempengaruhi PL berinteraksi dengan elemen sistem berbasis komputer.

Konsep sebuah interface diinterpretasi untuk menentukan:

1. Hardware yg mengeksekusi PL dan device yg dikontrol secara tidak langsung oleh PL
2. Software yg sudah ada dan harus dihubungkan dengan PL yg baru
3. Manusia yg menggunakan PL melalui keyboard atau perangkat I/O lain
4. Prosedur

Sumber Daya

1. Manusia
2. Perangkat Lunak Kategori yg diusulkan BEUNATAN
 - Komponen Off-the-self
 - Komponen Full-Experience
 - Komponen Partial-Experience
 - Komponen Baru
3. Lingkungan (Software Engineering Environment - SEE), menggabungkan PL dan Perangkat Keras.

Estimasi biaya dan usaha dapat dilakukan dengan cara :

1. Menunda estimasi sampai akhir proyek.
2. Berdasarkan estimasi pada proyek yg mirip sebelumnya.

3. Menggunakan 'teknik dekomposisi' yg relatif sederhana u/ estimasi biaya dan usaha proyek.
4. Menggunakan satu atau lebih model empiris bagi estimasi usaha dan biaya PL.

Akurasi estimasi proyek PL didasarkan pada :

1. Tingkat dimana perencana telah dengan tepat mengestimasi ukuran produk yg akan dibuat.
2. Kemampuan mengestimasi ukuran ke dalam kerja manusia, waktu kalender, dan dolar.
3. Tingkat dimana rencana proyek mencerminkan kemampuan tim PL.
4. Stabilitas syarat produk serta lingkungan yg mendukung usaha pengembangan PL.

Putnam dan Myers mengusulkan 4 masalah penentuan ukuran :

- Fuzzy-logic sizing (logika kabur)

Perencana harus mengidentifikasi tipe aplikasi, membuat besarannya dalam skala kuantitatif kemudian dibandingkan dengan rentang orisinal.

- Function point sizing

Perencana mengembangkan estimasi berdasarkan karakteristik domain informasi. -

Standard component sizing PL dibangun dari sejumlah 'komponen standar' yg umum (subsistem, modul, laporan, program interaktif).

- Change sizing

Digunakan jika PL yang ada harus dimodifikasi dengan banyak cara sebagai bagian dari proyek.

Data baris kode (LOC) dan titik fungsi (FP) pada estimasi proyek digunakan sbg :

1. variabel estimasi yg dipakai untuk mengukur masing-masing elemen PL.
2. metrik baseline yang dikumpulkan dari proyek yg lalu dan dipakai dengan variabel estimasi untuk mengembangkan proyeksi kerja dan biaya.

Expected Value untuk variabel estimasi :

$$EV = (S_{opt} + 4S_m + S_{pess}) / 6$$

EV = Expected value

S_{opt} = Estimasi optimistik

S_m = Estimasi paling sering

S_{pess} = Estimasi pesimistik

Apakah estimasi ini benar ? ' Kita tidak yakin!' Bagaimanapun canggih teknik estimasi harus di-cross-check dengan pendekatan lain.

Contoh estimasi berbasis LOC

PL CAD akan menerima data geometri dua dan tiga dimensi dari seorang perekayasa yang akan berinteraksi dan mengontrol sistem CAD melalui suatu interface pemakai. Kajian spesifikasi sistem menunjukkan bahwa PL akan mengeksekusi Workstation dan harus berinteraksi dengan berbagai peripheral grafis komputer spt mouse, digitizer dan printer laser.

Diketahui :

Perhitungan LOC untuk fungsi analisis geometri 3D (3DGA) :

optimis : 4600

most likely : 6900

pesimistik : 8600

$$EV = (4600 + 4 \cdot 6900 + 8600) / 6$$

$$= 6800 \text{ LOC}$$

Jumlah tersebut dimasukkan ke dalam tabel, begitu juga untuk perhitungan yang lain.

Sehingga diperoleh :

Tabel perkiraan (estimasi) untuk metode LOC

Fungsi	LOC terestimasi
interface pemakai & fasilitas kontrol (UICF)	2.300
analisis geometrik dua dimensi (2DGA)	5.300
analisis geometrik tiga dimensi (3DGA)	6.800
manajemen databse (DBM)	3.350
fasilitas display grafis komputer (CGDF)	4.950
kontrol peripheral (PC)	2.100
modul analisis desain (DAM)	8.400
baris kode terestimasi	33.200

Jika :

Produktifitas rata-rata organisasional = 620 LOC/person-month

Upah karyawan = \$8.000 per bulan

Biaya per baris kode = \$13

Maka :

$$\text{Tingkat produktifitas} = \frac{\text{jumlah titik fungsi}}{\text{jumlah orang-bulan}}$$

$$\text{Jumlah karyawan} = \frac{33200 \text{ LOC}}{620 \text{ LOC/bln}} = 53,5 \approx 54 \text{ orang}$$

$$\begin{aligned} \text{Estimasi biaya proyek berdasar LOC} \\ &= 33.200 \text{ LOC} * \$ 13 \\ &= \$ 431.600 \end{aligned}$$

$$\begin{aligned} \text{Estimasi biaya proyek berdasar upah} \\ &= 54 \text{ orang} * \$8.000 \\ &= \$432.000 \end{aligned}$$

Estimasi Berbasis FP (Function Point)

Dekomposisi untuk perhitungan berbasis FP berfokus pada harga domain info daripada fungsi PL. Perencana proyek memperkirakan input, output, inquiry, file dan interface eksternal. Untuk tujuan perkiraan tersebut faktor pembobotan kompleksitas diasumsikan menjadi rata-rata.

Setiap faktor pembobotan kompleksitas diestimasi dan faktor penyesuaian kompleksitas dihitung seperti dibawah ini :



Faktor	Harga
Backup and recovery	4
Komunikasi data	2
Pemrosesan terdistribusi	0
Kinerja kritis	4
Lingkungan operasi yang ada	3
Entri data on-line	4
Transaksi input pada layar ganda	5
File master yang diperbarui on-line secara on-line	3
Nilai kompleks domain informasi	5
Pemrosesan internal yang kompleks	5
Kode yg didesain untuk dapat dipakai lagi	4
Konversi/instalasi dalam disain	3
Instalasi ganda	5
Aplikasi yg didesain bagi perubahan	5
Faktor penyesuaian kompleksitas	1.17
TOTAL	53.17

Perkiraan harga domain informasi :

Tabel perkiraan harga domain informasi

nilai domain informasi	opt	likely	pess	jumlah estimasi	bobot	jumlah FP
jumlah input	20	24	30	24	4	96
jumlah output	12	15	22	16	5	80
jumlah inquiry	16	22	28	22	4	88
jumlah file	4	4	5	4	10	40
jumlah interface eksternal	2	2	3	2	7	14
jumlah total						318

jumlah estimasi (lihat rumus EV)

bobot (lihat kembali bab 4)

jumlah FP = jumlah estimasi * bobot

Total faktor pembobotan = $SF_i = 53.17$

Total FP = 318

FP terestimasi = jumlah total * $(0.65 + 0.01 * SF_i)$

= $318 * (0.65 + 0.01 * 53.17)$

= 375

Diketahui :

Produktifitas = 6.5 LOC/pm (dari historis)

Upah = \$ 8.000/m

Biaya FP = $\frac{\$ 8.000}{65 \text{ LOC}} = \$ 1.230$

Estimasi biaya proyek

= Biaya FP * FP terestimasi

= \$ 1.230 * 375

= \$ 461.250

Usaha terestimasi

= $\frac{\text{Total biaya}}{\text{upah/p}} = \frac{\$ 461.250}{\$ 8.000} = 58 \text{ p/m}$

Model Cocomo

Barry Boehm memperkenalkan hirarki model estimasi PL dengan nama COCOMO (COConstructive COSt Model = Model Biaya Konstruktif) yang berbentuk sbb :

1. Model COCOMO Dasar

Menghitung usaha pengembangan PL (dan biaya) sbg fungsi dari ukuran program yg diekspresikan dalam baris kode yg diestimasi (LOC).

2. Model COCOMO Intermediate

Menghitung usaha pengembangan PL sbg fungsi ukuran program dan serangkaian 'pengendali biaya' yg menyangkut penilaian yg subyektif thd produk, perangkat keras, personil dan atribut proyek.

3. Model COCOMO Advance

Menghubungkan semua karakteristik versi intermediate dg penilaian thd pengaruh pengendali biaya pd setiap langkah (analisis, perancangan, dll) dari proses rekayasa PL.

Model COCOMO mendefinisikan 3 kelas proyek PL yi :

1. Model Organik

Ukuran proyek relatif kecil, PL yang dibuat atau dikembangkan lebih simpel dengan aplikasi kerja yg baik. Misal program analisis termal yang dikembangkan untuk kelompok transfer panas.

2. Model Semi Detached

Ukuran proyek dan kekompleksan perangkat cukup besar dengan pengalaman kerja campuran (ada yg telah berpengalaman dan ada yg belum berpengalaman). Misal sistem pemrosesan transaksi dengan syarat tertentu untuk perangkat keras terminal dan perangkat lunak database.

3. Model Embedded

Ukuran proyek dan kekompleksan PL yg dikembangkan atau dikerjakan besar. Misal perangkat lunak kontrol penerbangan untuk pesawat udara.

Persamaan COCOMO Dasar

b_b

$E = a_b (KLOC)$

d_b

$D = c_b E$

Dimana : E = Effort (usaha yang diaplikasikan - pm)

D = waktu pengembangan (m)

KLOC = jumlah perkiraan baris kode (dalam ribuan)

a_b, b_b, c_b, d_b = koefisien (lihat tabel)

Tabel Model COCOMO Dasar

Proyek PL	a_b	b_b	c_b	d_b
organik	2.4	1.05	2.5	0.38
semi-detached	3.0	1.12	2.5	0.35
embedded	3.6	1.20	2.5	0.32

Model dasar ini dapat diperluas dengan mempertimbangkan kumpulan 'atribut pengendali biaya' yg dikelompokkan dalam 4 kategori utama :

1. Atribut produk

- ukuran keandalan proyek
- ukuran dari aplikasi database
- kekompleksan produk

2. Atribut perangkat keras

- kendala performansi run-time
- kendala memori
- lingkungan dari violability dari virtual memori
- waktu perputaran yg diperlukan

3. Atribut personil

- kemampuan sistem analis
- kemampuan software engineering
- pengalaman aplikasi
- pengalaman virtual mesin
- pengalaman bahasa pemrograman

4. Atribut proyek

- pemakaian alat bantu PL
- metode aplikasi software engineering
- jadwal pengembangan Masing-masing dari 15 atribut di atas dirata-rata dlm sebuah skala 6 titik dg rentang dari 'sangat rendah' ke 'sangat tinggi' (dlm kepentingan atau harga).

Persamaan COCOMO Intermediate

b_i

$$E = a_i (\text{KLOC})^{b_i} * \text{EAF}$$

dimana :

EAF = Effort Adjustment Factor (faktor penyesuaian usaha) yg mempunyai range antara 0.9 sampai 1.4

a_i, b_i = koefisien (lihat tabel)

Tabel Model COCOMO Intermediate

Proyek PL	a_i	b_i
organik	3.2	1.05
semi-detached	3.0	1.12
embedded	2.8	1.20

Contoh estimasi model COCOMO Kita aplikasikan model dasar pada contoh PL CAD sebelumnya dengan koefisien seperti pada tabel

bb

$$\begin{aligned}
 E &= ab \text{ (KLOC)} \\
 E &= 2.4 \text{ (KLOC)}^{1.05} \\
 &= 2.4 (33.2)^{1.05} \\
 &= 95 \text{ pm}
 \end{aligned}$$

Harga ini jauh lebih tinggi dibanding estimasi sebelumnya karena model COCOMO mengasumsikan tingkat LOC/pm yang jauh lebih rendah.

Untuk menghitung durasi proyek :

$$\begin{aligned}
 &\text{db} \\
 D &= cb E \\
 D &= 2.5 (E)^{0.38} \\
 &= 2.5 (95)^{0.38} \\
 &= 12.3 \text{ bulan}
 \end{aligned}$$

Harga durasi proyek memungkinkan perencana untuk menentukan jumlah orang yang disetujui (N)

$$\begin{aligned}
 N &= E/D \\
 &= 95/12.3 \\
 &= 7,7 \gg 8 \text{ orang}
 \end{aligned}$$

Kenyataannya, perencana dapat memutuskan hanya menggunakan 4 orang saja dan memperpanjang durasi proyek.

Catatan : Hubungan antara usaha dan waktu tidak linier.

Keputusan MAKE-BUY

Pada aplikasi PL, dari segi biaya sering lebih efektif membeli dari pada mengembangkan sendiri. Manajer RPL dihadapkan pada keputusan make-buy dengan pilihan :

1. PL dapat dibeli (atau lisensi) off-the-self.
2. Komponen PL full-experience dan partial-experience, dapat diperoleh dan kemudian dimodifikasi dan integrasi untuk memenuhi kebutuhan sendiri.
3. PL dapat dibuat custom-built oleh kontraktor luar untuk memenuhi spesifikasi pembeli.

Untuk produk PL yang mahal, langkah-langkah di bawah ini dapat dipertimbangkan:

1. Kembangkan spesifikasi untuk fungsi dan kinerja PL yg diperlukan.
2. Perkirakan biaya internal untuk pengembangan dan tanggal penyampaian.
- 3a. Pilih tiga atau empat calon aplikasi yang paling cocok dengan aplikasi anda. 3b. Pilih komponen yang reusable yg dapat membantu konstruksi aplikasi yg diperlukan.
4. Kembangkan sebuah matriks perbandingan untuk membandingkan calon PL.
5. Evaluasi masing-masing paket PL berdasarkan kualitas produk sebelumnya, dukungan penjual, arah proyek, reputasi dsb.
6. Hubungi pemakai PL lain dan mintalah pendapat mereka.

Pada analisis akhir, keputusan make-buy berdasarkan kondisi sbb:

1. Tanggal penyampaian
2. Biaya yang diperlukan
3. Dukungan

Membuat Pohon Keputusan

Rekayasa atau organisasi PL dapat menggunakan teknik statistik analisis pohon keputusan dengan pilihan :

1. membangun sistem X dari permulaan
2. menggunakan lagi komponen partial experience yang ada untuk membangun sistem
3. membeli sebuah produk perangkat lunak yang dapat diperoleh dan dimodifikasi untuk memenuhi kebutuhan lokal
4. mengkontrakkan pengembangan PL ke vendor luar

Bila sistem dibangun dari permulaan, hanya 70% probabilitasnya sehingga pekerjaan menjadi sulit. Perencana proyek dapat memproyeksikan usaha pengembangan yang sulit berbiaya \$450.000, usaha yang sederhana diperkirakan berbiaya \$380.000.

Expected value untuk biaya dihitung sepanjang cabang pohon keputusan, adalah :

$$\text{Expected Cost} = \sum (\text{jalur probabilitas})_i * (\text{biaya jalur terestimasi})_i$$

dimana i adalah garis edar pohon keputusan.

$$\text{Contoh : expected cost}_{\text{build}} = 0.30 (\$380 \text{ K}) + 0.70 (\$450 \text{ K}) = \$ 429 \text{ K}$$

$$\begin{aligned} \text{expected cost}_{\text{reuse}} &= 0.40 (\$275 \text{ K}) + 0.60 (0.20 (\$310 \text{ K}) + 0.80 (\$490 \text{ K})) \\ &= \$ 382 \text{ K} \end{aligned}$$

$$\text{expected cost}_{\text{buy}} = 0.70 (\$210 \text{ K}) + 0.30 (\$400 \text{ K}) = \$ 267 \text{ K}$$

$$\text{cost}_{\text{contract}} = 0.60 (\$350 \text{ K}) + 0.40 (\$500 \text{ K}) = \$ 410 \text{ K}$$

Berdasar biaya probabilitas dan proyeksi, expected cost yang paling rendah adalah pilihan buy

Catatan : Banyak kriteria yang harus dipertimbangkan, bukan hanya biaya, seperti pengalaman pengembang/ vendor/ kontraktor, penyesuaian

kebutuhan, kecenderungan perubahan dapat mempengaruhi keputusan akhir!

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S. Pressman

MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Spesifikasi perangkat lunak.
Kajian spesifikasi perangkat lunak.

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka
06

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Bahasan mengenai spesifikasi perangkat lunak yang dipergunakan untuk memenuhi kebutuhan pemakai.

Kompetensi

Mahasiswa dapat menjabarkan spesifikasi Perangkat Lunak dan melakukan pengkajian spesifikasi Perangkat Lunak

Spesifikasi Perangkat Lunak

Definisi

Kebutuhan Perangkat Lunak adalah kondisi, kriteria, syarat atau kemampuan yang harus dimiliki oleh perangkat lunak untuk memenuhi apa yang disyaratkan atau diinginkan pemakai.

Jenis perangkat lunak :

1. Kebutuhan fungsional : kebutuhan yang berkaitan dengan fungsi atau proses transformasi yang harus mampu dikerjakan oleh perangkat lunak.
contoh : perangkat lunak harus dapat menyimpan semua rincian data pesanan pelanggan
2. Kebutuhan Antarmuka : kebutuhan yang menghubungkan perangkat lunak dengan elemen perangkat keras, perangkat lunak, atau basis data.
contoh : perangkat untuk input data dapat berupa keyboard, mouse, dan scanner.
3. Kebutuhan unjuk kerja : kebutuhan yang menetapkan karakteristik unjuk kerja yang harus dimiliki oleh perangkat lunak
contoh : perangkat lunak harus bisa mengolah data sampai 1 juta record untuk tiap transaksi

1. Pendahuluan

Dalam Pembuatan Spesifikasi Kebutuhan Perangkat Lunak (SKPL) yang dapat menggambarkan kebutuhan pengguna dan memberikan arah agar perancangannya sesuai dengan rancangan.

Tujuan

Membahas mengenai perangkat lunak yang kebutuhan software-nya ada pada dokumen ini. Gambarkan lingkup dari produk yang dilingkupi oleh SKPL ini, khususnya jika gambaran SKPL hanya bagian dari sebuah sistem atau sub sistem

Ruang Lingkup Perangkat Lunak

Memberikan gambaran singkat mengenai perangkat lunak yang akan dikembangkan, termasuk keuntungan, tujuan dan sasaran. Terangkan juga hubungan perangkat lunak dengan sasaran perusahaan atau strategi bisnis

Target Audience

Untuk target audience menjelaskan siapa yang harus memahami dan menggunakan dokumen. Dapat menunjukkan bagaimana perbedaan dari masing-masing audience tersebut dalam memperlakukan dokumen .

2. Deskripsi Umum

Tentang Perangkat Lunak

Menggambarkan secara apa adanya keadaan perangkat lunak yang ditetapkan dalam SKPL. Sebagai contoh, perangkat lunak yang mengganti sebagian sistem yang ada. Jika SKPL mendefinisikan komponen dari sistem yang besar, kebutuhan dari sistem tersebut di bagi menjadi dua, yaitu fungsionalitas perangkat lunak dan identifikasi antarmuka. Diagram sederhana dapat membantu untuk menunjukkan komponen utama dari sistem keseluruhan, interkoneksi subsistem dan antarmuka eksternal.

Fungsi-fungsi Perangkat Lunak

Berisi ringkasan dari fungsi-fungsi utama perangkat lunak. Dapat berupa ringkasan yang bersifat high level . Aturlah fungsi-fungsi tersebut agar mudah dipahami saat membaca SKPL . Gambarkan dengan menggunakan seperti data flow diagram level 0 atau object class diagram.

Karakteristik dan Klasifikasi Pengguna

Memperkenalkan variasi klasifikasi pengguna yang akan mempergunakan perangkat lunak ini. Klasifikasi pengguna bisa dibedakan berdasarkan banyak pengguna, kumpulan pengguna fungsi perangkat lunak, keahlian teknis, keamanan atau pembagian hak akses. Terangkan karakteristik dan kebutuhan yang berhubungan dengan setiap klasifikasi pengguna. Membedakan klasifikasi pengguna yang sangat penting untuk perangkat lunak ini, dengan klasifikasi pengguna yang dianggap kurang penting untuk dipuaskan.

Lingkungan Operasi

Merupakan gambaran lingkungan dimana perangkat lunak ini akan beroperasi, termasuk platform perangkat keras, versi dan sistem operasi, dan berbagai software atau aplikasi lain yang diperlukan untuk mendampinginya.

Batasan Desain dan Implementasi

Menggambarkan beberapa item atau isu yang dapat membatasi pengembangan perangkat lunak. Hal ini termasuk: kebijakan regulasi perusahaan, keterbatasan perangkat keras (timing requirements, memory requirements), antarmuka pada aplikasi lain, teknologi tertentu, tools, dan database yang digunakan, operasi paralel, kebutuhan bahasa, protokol komunikasi, pertimbangan keamanan, konvensi desain atau standart pemrograman (contohnya jika organisasi customer/pengguna akan bertanggung jawab dalam pemeliharaan perangkat lunak yang telah diberikan)

Dokumentasi Bagi Pengguna

Berisi daftar komponen-komponen dokumentasi yang diperuntukkan kepada pengguna (seperti: user manual, bantuan on-line, dan tutorials) dan yang akan diberikan bersama-sama perangkat lunaknya.

Asumsi dan Ketergantungan

Berisi daftar beberapa asumsi yang akan mempengaruhi beberapa kebutuhan dalam SKPL. Termasuk di dalamnya third-party atau komponen komersil dalam perencanaan yang digunakan, isu-isu atau batasan tentang pengembangan atau lingkungan operasi. Pekerjaan pembuatan SKPL ini akan terpengaruh jika asumsi-asumsinya tidak benar, tidak *shared*, atau berubah. Tunjukkan juga ketergantungan terhadap faktor eksternal, seperti komponen-komponen perangkat lunak yang dimaksudkan untuk digunakan pada proyek lain, kecuali kalau telah disiapkan pada dokumen lain (misalnya dalam dokumen perencanaan proyek).

3. Kebutuhan Antarmuka Eksternal

Antarmuka Pengguna

menguraikan karakteristik logik dari setiap antarmuka antara produk perangkat lunak dan penggunanya. Bisa berupa contoh gambar *screen*, beberapa standar GUI atau arahan bentuk yang harus diikuti, batasan *screen layout*, standart buttons dan function (misal help) yang akan kelihatan pada setiap *screen*, *keyboard shortcuts*, standart tampilan *error message*, dan yang lainnya. Tentukan komponen perangkat lunak yang diperlukan untuk antarmuka pengguna. Detail dari desain antarmuka pengguna ada pada dokumen terpisah yaitu spesifikasi antarmuka pengguna.

Antarmuka Perangkat Keras

Gambarkan karakteristik logik dan fisik dari setiap antarmuka antara produk perangkat lunak dan komponen perangkat keras dari sistem. Boleh berupa tipe peralatan pendukung, data alamiah dan kontrol interaksi antara perangkat lunak dan perangkat keras, dan protokol komunikasi yang digunakan.

Antarmuka Perangkat Lunak

menjelaskan koneksi antara perangkat lunak ini dengan komponen perangkat lunak tertentu lainnya (nama dan versi), termasuk database, sistem operasi, tools, libraries, dan komponen komersial yang terintegrasi. Tunjukkan item-item data atau pesan yang datang kepada sistem dan hasilnya dan gambaran dari penggunaan setiap hasil tersebut. Gambaran kebutuhan servis dan komunikasi. Menunjuk pada dokumen yang menguraikan detail pemrograman aplikasi interface protocol. Identifikasi data yang akan dibagi antar komponen perangkat lunak. Jika mekanisme pembagian data harus terimplementasi dengan cara yang khusus (contoh, penggunaan lingkungan data global sistem operasi multitasking), terutama batasan implementasinya.

Antarmuka Komunikasi

Menguraikan asosiasi kebutuhan dengan beberapa fungsi komunikasi yang dibutuhkan oleh perangkat lunak ini, termasuk e-mail, web browser, protokol komunikasi network server, forms elektronik, dan lain sebagainya. Identifikasi beberapa hal yang berhubungan dengan format message. Identifikasi beberapa standart komunikasi yang akan digunakan, seperti FTP atau HTTP. Menetapkan keamanan komunikasi atau isu tentang encrypsi, kecepatan transfer data, dan mekanisme sinkronisasi.

4. Feature Sistem

Adalah bagian untuk mengilustrasikan kebutuhan fungsional perangkat lunak dengan mengelompokkan secara feature sistem, yaitu servis utama yang disediakan oleh perangkat lunak. Pengelompokan fitur sistem pada bab ini sebaiknya dengan use case, jenis operasi, user class, object class, hirarki fungsionalitas atau

kombinasinya, apapun yang membuat dapat lebih mengetahui tentang perangkat lunak tersebut.

5. Kebutuhan NonFungsional Lainnya

Kebutuhan Kinerja

Jika ada kebutuhan kinerja perangkat lunak yang kondisinya bervariasi, nyatakan dan terangkan dasar pemikirannya, agar dapat membantu pengembang dalam memahami tujuan dan pemilihan desain yang cocok. Terutama yang berhubungan dengan waktu untuk sistem real time. Buatlah kebutuhan yang sedemikian jelas dan mungkin. Pernyataan kebutuhan kinerja untuk satu kebutuhan fungsional atau feature.

Kebutuhan Keamanan

Spesifikasikan kebutuhan yang mementingkan kemungkinan hilang, rusak atau kesalahan akan hasil dari penggunaan perangkat lunak. Tentukan beberapa usaha perlindungan atau aksi yang harus dilakukan untuk mencegahnya. Tunjukkan beberapa kebijakan eksternal atau regulasi isu tentang keamanan yang mempengaruhi penggunaan dan desain perangkat lunak. Temukan beberapa sertifikasi keamanan yang dapat memberikan kepuasan.

Kebutuhan Perlindungan Keamanan

Spesifikasikan kebutuhan yang concern pada keamanan atau isu privasi di sekitar penggunaan perangkat lunak atau proteksi oleh perangkat lunak pada penggunaan atau pembuatan data. Tentukan kebutuhan autentifikasi identitas pengguna. Tunjukkan beberapa kebijakan eksternal atau regulasi yang berisi isu-isu keamanan yang mempengaruhi penggunaan perangkat lunak. Temukan beberapa sertifikasi keamanan atau privasi yang harus memuaskan.

Atribut Kualitas Perangkat Lunak

Spesifikasikan beberapa tambahan karakteristik kualitas dari perangkat lunak yang penting bagi pengguna atau pengembang. Pertimbangkan tentang adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability dan usability. Tuliskan pertimbangan-

pertimbangan tersebut agar menjadi spesifik, kuantitatif dan memungkinkan untuk diverifikasi. Setidaknya, klarifikasikan preferensi relatif dari variasi atribut, seperti lebih mudah menggunakannya dari pada mempelajarinya.

Aturan Penggunaan

Daftar beberapa prinsip pengoperasian perangkat lunak, seperti fungsi-fungsi yang dapat dilakukan seseorang pada situasi tertentu. Ingat, bukan untuk kebutuhan fungsional, tetapi yang menyatakan beberapa kebutuhan fungsional tertentu sebagai sebuah aturan.

6. Kebutuhan Lain

Tentukan beberapa kebutuhan lain yang tidak tercover pada SKPL ini. Mungkin bisa termasuk kebutuhan database, kebutuhan menginternasionalisasikan, kebutuhan legal/hukum, penggunaan kembali pada sebuah proyek, dan sebagainya. Ditambah beberapa bagian yang relevan untuk SKPL tersebut.

Model Analisis

Dapat digambarkan dengan menggunakan model analisis seperti data flow diagram, class diagram, state-transition diagram, atau entity relationship diagram.



Daftar Kebutuhan

Berisi daftar nomer-nomer kebutuhan yang dapat ditunjukkan pada SKPL, sehingga dapat ditelusuri asalnya.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman
- IEEE Std. 1233, 1998 Edition IEEE Guide for Developing System Requirements Specifications
- IEEE, Software Requirements Engineering, Second Edition, IEEE Computer Society Press, 2002.
- Bray, Ian K. An Introduction to Requirement Engineering, 1st published, Addison-Wesley, 2002
- Kotonya, Gerald and Sommerville, Ian. Requirement Engineering: Processes and Techniques, John Wiley & Sons Ltd, 1998



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Elemen model analisis.
Permodelan data, fungsional,
aliran informasi dan tingkah
laku. Mekanisme dari analisis
terstruktur.

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

Kode MK
87011

Disusun Oleh
Tim Dosen

07

Abstract

Petunjuk Penggunaan Template
Modul Standar untuk digunakan
dalam modul perkuliahan
Universitas Mercu Buana

Kompetensi

Dosen Penyusun dapat menerapkan
dan menggunakan template modul
standar untuk modul-modul yang
akan dipergunakannya

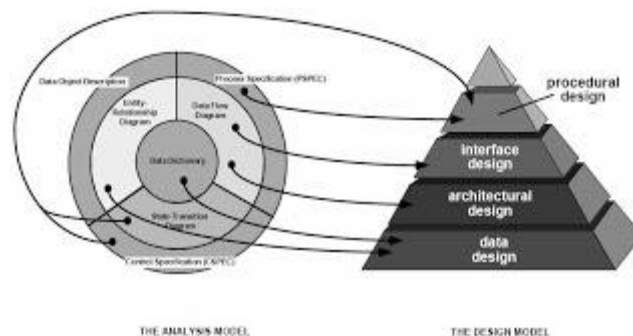
Pemodelan Analisis

Pada tingkat teknik, rekayasa perangkat lunak dimulai dengan serangkaian tugas pemodelan yang membawa kepada suatu spesifikasi lengkap dari persyaratan representasi dan representasi desain yang komprehensif bagi perangkat lunak yang dibangun.

a. Elemen Model Analisis

Model analisis harus dapat mencapai tiga sasaran utama yakni untuk :

- Menggambarkan apa yang dibutuhkan untuk pelanggan
- Membangun dasar bagi pembuatan desain perangkat lunak
- Membatasi serangkaian persyaratan yang dapat divalidasi begitu perangkat lunak dibangun.



Untuk mencapai sasaran tersebut dibuatlah model analisis yang berisi:

- Data Dictionary

Penyimpanan yang berisi deskripsi dari semua obyek data yang dikonsumsi atau diproduksi oleh perangkat lunak.

- Entity Relationship Diagram (ERD)

Menggambarkan hubungan antara obyek data.

- Data Flow Diagram (DFD)

- o Memberikan indikasi mengenai bagaimana data ditransformasi pada saat data bergerak melalui sistem
- Menggambarkan fungsi-fungsi (dan sub fungsi) yang mentransformasikan aliran data.

- State Transition Diagram

Menunjukkan bagaimana sistem bertindak laku sebagai akibat dari kejadian eksternal.

- Control Specification (CSPEC)

Informasi tambahan mengenai aspek kontrol dari perangkat lunak

b. Pemodelan Data

Pemodelan data merupakan sebuah tahapan dalam merancang sebuah sistem informasi. Pemodelan data berfokus pada data apa yang akan disimpan yang menggambarkan hubungan antara entiti set yang dibutuhkan oleh sebuah organisasi dalam pengelolaan data.

Untuk dapat menjawab tentang pemodelan data sebagai berikut :

1. Bagaimana komposisi dari masing-masing obyek data dan atribut apa yang menggambarkan obyek tersebut?
2. Dimana obyek saat ini berada?
3. Bagaimana hubungan antara masing-masing obyek data dan obyek lainnya?
4. Bagaimana hubungan antara obyek dengan proses yang mentransformasikannya?

Entity Relational Diagram (ERD)

1. Obyek Data, Atribut dan Hubungan

Obyek Data Adalah representasi dari hamper semua informasi gabungan yang harus dipahami oleh perangkat lunak. Atribut Menentukan property suatu obyek data dan mengambil salah satu dari tiga karakteristik yang berbeda.

- o Menamai sebuah contoh dari obyek data
 - o Menggambarkan contoh
 - o Membuat referensi ke contoh yang lain pada tabel yang lain.
- Hubungan Obyek data disambungkan satu dengan lainnya dengan berbagai macam cara.

2. Kardinalitas dan Modalitas Kardinalitas

Model data harus dapat merepresentasikan jumlah peristiwa dari obyek di dalam hubungan yang diberikan

- Satu ke satu (1:1) Misalnya: seorang suami hanya dapat memiliki satu istri, dan seorang istri hanya mempunyai satu suami
- Satu ke banyak (1:N) Misalnya: seorang ibu dapat memiliki banyak anak, tetapi seorang anak hanya dapat memiliki satu ibu
- Banyak ke banyak (M:N) Misalnya: seorang paman dapat memiliki banyak keponakan, sementara itu seorang keponakan dapat memiliki banyak paman
-

Modalitas Modalitas dari suatu hubungan adalah nol bila tidak ada kebutuhan eksplisit untuk hubungan yang terjadi atau hubungan itu bersifat opsional. Modalitas bernilai satu jika suatu kejadian dari hubungan merupakan perintah.

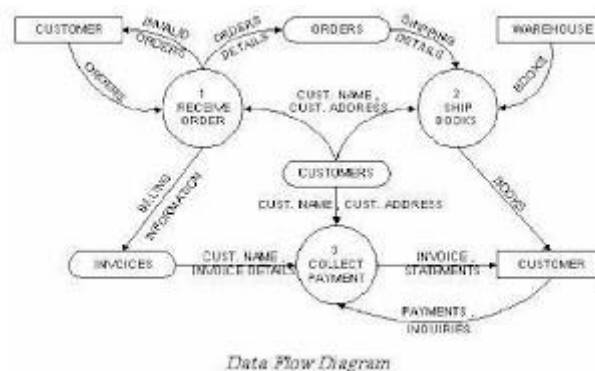
Entity Relational Diagram pada mulanya digunakan untuk desain sistem database relational dan telah dikembangkan oleh yang lainnya. Serangkaian komponen utama diidentifikasi untuk ERD: obyek data, atribut, hubungan dan berbagai tipe indicator. Tujuan utama dari ERD adalah untuk mewakili obyek data dan hubungan mereka.

c. Pemodelan Fungsional dan Aliran Informasi

Informasi ditransformasikan pada saat dia mengalir melalui sebuah sistem berbasis komputer. Sistem tersebut menerima input dengan berbagai cara dan menghasilkan suatu output. Akibatnya kita dapat menciptakan suatu model aliran bagi setiap sistem berbasis komputer tanpa melihat ukuran dan kompleksitasnya.

1. Diagram Aliran Data/ Data Flow Diagram (DFD)

Merupakan sebuah teknik grafis yang menggambarkan aliran informasi dan transformasi yang diaplikasikan pada saat data bergerak dari input menjadi output. Dikenal juga dengan sebutan grafik aliran data atau bubble chart.



Komponen-komponen DFD :

- Proses : menunjukkan apa yang dikerjakan oleh sistem. Setiap proses memiliki nama yang unik dan nomor yang ditempatkan dalam simbol.
- External entity adalah di luar sistem, tetapi mereka merupakan salah satu bagian yang memberikan input data ke dalam sistem atau digunakan oleh output sistem
- Data Flow adalah tempat penyimpanan data
- Data Store : Proses dapat menempatkan data ke dalam data store atau mengambil / mendapatkan data store. Setiap data store mempunyai nama yang unik External Entity

Pemodelan Tingkah Laku :

Model perilaku menggambarkan bagaimana PL merespon event atau stimulan eksternal. Untuk model tersebut, analisis harus melakukan langkah-langkah sebagai berikut :

- Evaluasi semua use-case untuk mendapatkan pemahaman menyeluruh tentang urutan interaksi di dalam sistem.
- Mengenali event yang mengendalikan urutan interaksi dan memahami bagaimana event mempunyai relasi terhadap objek spesifik.
- Membuat urutan untuk setiap use-case.
- Membangun state diagram untuk sistem.
- Review model behavioral untuk memverifikasi akurasi dan konsistensi

Mekanik dari analisis terstruktur

Dalam konteks pemodelan perilaku, dua karakter keadaan harus diperhatikan:

- Keadaan setiap class ketika sistem menjalankan fungsinya, dan
- Keadaan sistem ketika diobservasi dari luarsebagaimana sistem menjalankan fungsinya.

Keadaan class mengambil baik karakter aktif maupun pasif.

- Sebuah keadaan pasif adalah status saat ini dari semua atribut objek.
- Keadaan aktif dari sebuah objek menggambarkan status saat ini pada objek tersebut ketika menjalankan transformasi atau proses.

Kamus Data

Kamus data adalah suatu daftar data elemen yang terorganisir dengan definisi yang tetap dan sesuai dengan sistem, sehingga user dan analisis sistem mempunyai

pengertian yang sama tentang input, output, dan komponen data store. Kamus data ini sangat membantu analis sistem dalam mendefinisikan data yang mengalir di dalam sistem, sehingga pendefinisian data itu dapat dilakukan dengan lengkap dan terstruktur. Pembentukan kamus data dilaksanakan dalam tahap analisis dan perancangan suatu sistem. Pada tahap analisis, kamus data merupakan alat komunikasi antara user dan analis sistem tentang data yang mengalir di dalam sistem, yaitu tentang data yang masuk ke sistem dan tentang informasi yang dibutuhkan oleh user. Sementara itu, pada tahap perancangan sistem kamus data digunakan untuk merancang input, laporan dan database. Pembentukan kamus data didasarkan atas alur data yang terdapat pada DFD. Alur data pada DFD ini bersifat global, dalam arti hanya menunjukkan nama alur datanya tanpa menunjukkan struktur dari

Metode Analisis Klasik

Model Proses Perangkat Lunak Evolusioner

Model evolusioner adalah model iteratif. Model itu ditandai dengan tingkah laku yang memungkinkan perekayasa perangkat lunak mengembangkan versi perangkat lunak yang lebih lengkap sedikit demi sedikit.

Model Pertambahan

Model inkremental menggabungkan elemen-elemen model sekuensial linier dengan filosofi prototipe iteratif. Pada saat model pertambahan dipergunakan, pertambahan pertama sering merupakan produk inti (core product), yaitu sebuah model pertambahan yang dipergunakan, tetapi beberapa muka tambahan tetap tidak disampaikan. Produk inti tersebut dipergunakan oleh pelanggan (atau mengalami pengkajian lebih detail). Sebagai hasil dari pemakaian dan/atau evaluasi, maka dikembangkan rencana bagi pertambahan selanjutnya. Rencana tersebut menekankan modifikasi produk inti untuk secara lebih baik memenuhi kebutuhan para pelanggan dan penyampaian fitur serta fungsionalitas tambahan. Proses ini diulang mengikuti penyampaian setiap pertambahan sampai bisa menghasilkan produk yang lengkap. Model proses pertambahan tersebut, seperti model prototipe dan pendekatan-pendekatan evolusioner yang lain, bersifat iteratif. Tetapi tidak seperti model prototipe, model pertambahan berfokus pada penyampaian produk operasional dalam setiap pertambahannya. Pertambahan awal ada di versi stripped down dari produk akhir, tetapi memberikan kemampuan untuk melayani pemakai dan juga menyediakan platform untuk evaluasi oleh pemakai.

Model Spiral

Model spiral yang pada awalnya diusulkan oleh Boehm adalah model proses perangkat lunak yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier. Model itu berpotensi untuk pengembangan versi pertambahan perangkat lunak secara cepat.

- Model spiral dibagi menjadi sejumlah aktivitas kerangka kerja, disebut juga wilayah tugas, di antara tiga sampai enam wilayah tugas. Gambar 2.8. menggambarkan model spiral yang berisi enam wilayah tugas : Komunikasi pelanggan – tugas-tugas yang dibutuhkan untuk membangun komunikasi yang efektif di antara pengembang dan pelanggan
- Perencanaan – tugas-tugas yang dibutuhkan untuk mendefinisikan sumber-sumber daya, ketepatan waktu, dan proyek informasi lain yang berhubungan.
- Analisis risiko – tugas-tugas yang dibutuhkan untuk menaksir risiko-risiko, baik manajemen maupun teknis.
- Perekrutan – tugas-tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut
- Konstruksi dan peluncuran – tugas-tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (install) dan memberikan pelayanan kepada pemakai (contohnya pelatihan dan dokumentasi).
- Evaluasi pelanggan – tugas-tugas yang dibutuhkan untuk memperoleh umpan balik dari pelanggan dengan didasarkan pada evaluasi representasi perangkat lunak, yang dibuat selama masa perekrutan, dan diimplementasikan selama masa pemasangan.

Ketika proses evolusioner ini mulai, tim rekayasa perangkat lunak bergerak searah jarum mengelilingi spiral tersebut dengan dimulai intinya. Lintasan pertama putaran spiral menghasilkan perkembangan dari spesifikasi produk; putaran spiral selanjutnya mungkin dipakai untuk mengembangkan sebuah prototipe, dan secara progresif mengembangkan versi perangkat lunak yang lebih canggih. Tidak seperti model proses klasik yang berakhir pada saat perangkat lunak sudah disampaikan, model spiral bisa disesuaikan agar perangkat lunak bisa dipakai selama hidup perangkat lunak komputer.

Model spiral menjadi sebuah pendekatan yang realistis bagi perkembangan sistem dan perangkat lunak skala besar. Karena perangkat lunak terus bekerja selama proses bergerak, pengembang dan pemakai memahami dan bereaksi lebih baik terhadap risiko dari setiap tingkat evolusioner. Model spiral menggunakan prototipe sebagai mekanisme pengurangan risiko.

Model Rakitan Komponen

Model rakitan komponen menggabungkan beberapa karakteristik model spiral. Model ini bersifat evolusioner, sehingga membutuhkan pendekatan iteratif untuk menciptakan perangkat lunak. Tetapi model rakitan komponen merangkai aplikasi dari komponen perangkat lunak sebelum dipaketkan (kadang-kadang disebut “kelas”). Model rakitan komponen membawa kepada penggunaan kembali perangkat lunak, dan kegunaan kembali tersebut memberi sejumlah keuntungan yang bisa diukur pada perekayasa perangkat lunak.

Model perkembangan Konkuren

Model proses konkuren sering digunakan sebagai paradigma bagi pengembangan aplikasi klien/server. Sistem klien/server dirancang dari serangkaian komponen fungsional. Bila diaplikasikan kepada klien/server, model proses konkuren akan mendefinisikan aktivitas di dalam dua dimensi : dimensi sistem, dan dimensi komponen. Isu tingkat sistem ditujudengan menggunakan tiga aktivitas : desain, assembly, dan pemakaian. Dimensi komponen dituju dengan dua aktivitas : desain dan rea-lisasi. Konkuren dicapai dengan dua cara :

1. aktivitas sistem dan komponen yang berlangsung secara simultan dan dapat dimodelkan dengan menggunakan pendekatan orientasi keadaan yang digambarkan di atas;
2. aplikasi klien/server khusus diimplementasikan dengan banyak komponen di mana masing-masing bisa dirancang dan direalisasikan secara konkuren.

Kenyataannya model proses konkuren bisa diaplikasikan ke dalam semua tipe perkembangan perangkat lunak, dan memberikan gambaran akurat mengenai keadaan tertentu dari sebuah proyek.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Pemodelan data (Object Data dan Entity Relationship Diagram)

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka
09

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Membuat DFD, ERD, Spesifikasi proses dan Data Dictionary yang dibutuhkan dalam pembuatan RPL

Kompetensi

Mahasiswa dapat membuat Data Flow Diagram, Entity Relationship Diagram

Pemodelan Data

Pemodelan Data dalam rekayasa perangkat lunak adalah proses menciptakan sebuah model data dengan menerapkan model deskripsi formal data menggunakan teknik pemodelan data. Pemodelan data adalah metode yang digunakan untuk menentukan dan menganalisis persyaratan data yang diperlukan untuk mendukung proses bisnis suatu organisasi. Data yang dibutuhkan adalah dicatat sebagai data model konseptual dengan definisi data yang terkait. Realisasi penerapan model konseptual yang disebut model data logis. Untuk menerapkan satu model konseptual data mungkin membutuhkan beberapa model data logis. Pemodelan data mendefinisikan elemen tidak hanya data, tapi struktur dan hubungan antara mereka teknik pemodelan data dan metodologi yang digunakan untuk model data dengan cara yang standar yang konsisten, dapat diprediksi untuk mengelolanya sebagai sumber daya.

Pada tingkat teknik, rekayasa perangkat lunak dimulai dengan serangkaian tugas pemodelan. Model analisis sebenarnya merupakan serangkaian model yang merupakan representasi teknis yang pertama dari system. Di dalam suatu industri dikenal berbagai macam proses, demikian juga halnya dengan industri perangkat lunak. Perbedaan proses yang digunakan akan menguraikan aktivitas-aktivitas proses dalam cara-cara yang berlainan. Perusahaan yang berbeda menggunakan proses yang berbeda untuk menghasilkan produk yang sama. Tipe produk yang berbeda mungkin dihasilkan oleh sebuah perusahaan dengan menggunakan proses yang berbeda. Namun beberapa proses lebih cocok dari lainnya untuk beberapa tipe aplikasi. Jika proses yang salah digunakan akan mengurangi kualitas kegunaan produk yang dikembangkan.

Karena banyaknya variasi dalam model proses yang digunakan maka tidak mungkin menghasilkan gambaran-gambaran yang reliabel untuk alokasi biaya dalam aktivitas-aktivitas ini.

Modifikasi perangkat lunak biasanya lebih dari 60 % dari total biaya pembuatan perangkat lunak. Presentasi ini terus bertambah karena lebih banyak perangkat lunak dihasilkan dan dipelihara. Pembuatan perangkat lunak untuk suatu perubahan adalah penting. Proses perangkat lunak kompleks dan melibatkan banyak aktivitas.

Seperti produk, proses juga memiliki atribut dan karakteristik seperti :

- Understandability, yaitu sejauh mana proses secara eksplisit ditentukan dan bagaimana kemudahan definisi proses itu dimengerti.
- Visibility, apakah aktivitas-aktivitas proses mencapai titik akhir dalam hasil yang jelas sehingga kemajuan dari proses tersebut dapat terlihat nyata/jelas
- Supportability, yaitu sejauh mana aktivitas proses dapat didukung oleh CASE
- Acceptability, apakah proses yang telah ditentukan oleh insinyur dapat diterima dan digunakan dan mampu bertanggung jawab selama pembuatan produk perangkat lunak
- Reliability, apakah proses didesain sedikikan rupa sehingga kesalahan proses dapat dihindari sebelum terjadi kesalahan pada produk.
- Robustness, dapatkah proses terus berjalan walaupun terjadi masalah yang tak diduga
- Maintainability, dapatkah proses berkembang untuk mengikuti kebutuhan atau perbaikan
- Rapidity, bagaimana kecepatan proses pengiriman sistem dapat secara lengkap memenuhi spesifikasi.

Tetapi pada saat ini ada dua landskap pemodelan analisis. Yaitu yang pertama **analisis terstruktur** adalah metode pemodelan klasik. Dimana analisis terstruktur ini merupakan aktifitas pembangunan model. Dan yang kedua adalah **analisis berorientasi Objek** . Tetapi pada makalah ini yang dijelaskan adalah Tinjauan singkat terhadap metode analisis yang umum digunakan. Untuk menciptakan model yang menggambarkan muatan dan aliran informasi (data dan kontrol).

Model

Tidak mungkin untuk mengoptimalkan semua atribut proses secara serentak. Contohnya, jika mengembangkan proses cepat dilakukan mungkin kita perlu mengurangi visibility proses karena pembuatan proses yg nyata berarti pembuatan dokumen secara teratur. Ini akan memperlambat proses.

Model proses perangkat lunak masih menjadi object penelitian, tapi sekarang ada banyak model umum atau paradigma yang berbeda dari pengembangan perangkat lunak, antara lain:

- Pendekatan Waterfall

Berisi rangkaian aktivitas proses seperti yang telah diuraikan diatas dan disajikan dalam proses yang terpisah, seperti spesifikasi kebutuhan, implementasi desain

perangkat lunak, uji coba dst. Setelah setiap langkah didefinisikan, langkah tersebut di *sign off* dan pengembangan dilanjutkan pada langkah berikutnya.

- Pengembangan secara evolusioner

Pendekatan ini interleaves aktivitas spesifikasi, pengembangan dan validasi. Sistem awal dengan cepat dikembangkan dari pelanggan untuk memproduksi sistem yang memenuhi kebutuhan pelanggan. Kemudian sistem disampaikan. Sistem itu mungkin diimplementasikan kembali dengan pendekatan yang lebih terstruktur untuk menghasilkan sistem yang kuat dan maintainable.

- Transformasi formal

Pendekatan ini berdasarkan pembuatan spesifikasi sistem formal secara matematik dan transformasi spesifikasi dengan menggunakan metode matematik atau dengan suatu program. Transformasi ini adalah *correctnesspreserving* ini berarti bahwa kita dapat yakin program yang dikembangkan sesuai dengan spesifikasi.

- Penggabungan sistem dengan menggunakan komponen-komponen yang dapat digunakan kembali.

Teknik ini menganggap bagian-bagian dari sistem sudah ada. Proses pengembangan sistem lebih berfokus pada penggabungan bagian-bagian daripada pengembangan tiap bagian.

Dua pertama dari pendekatan-pendekatan diatas yaitu waterfall dan pengembangan evolusioner, saat ini banyak digunakan dalam pengembangan sistem. Beberapa sistem sudah dibuat dengan menggunakan transformasi *correctness preserving* tapi ini masih menjadi penelitian.

Metode penggunaan kembali (*reuse*) umum di Jepang. Metode ini sekiranya akan diakui oleh Eropa dan Amerika Utara. Di US metode ini dimulai 1995 dengan anggaran 150 million dollars. Bagaimanapun juga reuse masih suatu penelitian, terlalu cepat untuk berkomentar tentang keefektifannya.

Waterfall

Model ini telah diperoleh dari proses engineering lainnya. Model ini menawarkan cara pembuatan perangkat lunak secara lebih nyata.

Langkah-langkah yang penting dalam model ini adalah

- Penentuan dan analisis spesifikasi

Jasa, kendala dan tujuan dihasilkan dari konsultasi dengan pengguna sistem. Kemudian semuanya itu dibuat dalam bentuk yang dapat dimengerti oleh user dan staf pengembang.

- Desain sistem dan perangkat lunak

Proses desain sistem membagi kebutuhan-kebutuhan menjadi sistem perangkat lunak atau perangkat keras. Proses tersebut menghasilkan sebuah arsitektur sistem keseluruhan. Desain perangkat lunak termasuk menghasilkan fungsi sistem perangkat lunak dalam bentuk yang mungkin ditransformasi ke dalam satu atau lebih program yang dapat dijalankan.

- Implementasi dan ujicoba unit

Selama tahap ini desain perangkat lunak disadari sebagai sebuah program lengkap atau unit program. Uji unit termasuk pengujian bahwa setiap unit sesuai spesifikasi.

- Integrasi dan ujicoba sistem

Unit program diintegrasikan dan diuji menjadi sistem yang lengkap untuk menyakinkan bahwa persyaratan perangkat lunak telah dipenuhi. Setelah ujicoba, sistem disampaikan ke pelanggan

- Operasi dan pemeliharaan

Normalnya, ini adalah phase yang terpanjang. Sistem dipasang dan digunakan.

Pemeliharaan termasuk pembetulan kesalahan yang tidak ditemukan pada langkah sebelumnya. Perbaikan implementasi unit sistem dan peningkatan jasa sistem sebagai kebutuhan baru ditemukan.

Dalam prakteknya, setiap langkah sering tumpang tindih dan saling memberi informasi satu sama lain. Proses perangkat lunak tidak linier dan sederhana tapi mengandung urutan iterasi dari aktivitas pengembangan. Selama di langkah terakhir, perangkat lunak telah digunakan. Kesalahan dan kelalaian dalam menentukan kebutuhan perangkat lunak original dapat diatasi.

Sayangnya, model yang banyak mengandung iterasi sehingga membuat sulit bagi pihak manajemen untuk memeriksa seluruh rencana dan laporan. Maka dari itu, setelah sedikit iterasi, biasanya bagian yang telah dikembangkan akan dihentikan dan dilanjutkan dengan langkah pengembangan selanjutnya. Masalah-masalah selama resolusi selanjutnya, dibiarkan atau diprogram. Pemberhentian yang prematur dari persyaratan akan berarti bahwa sistem tidak akan sesuai dengan keinginan user. Mungkin juga sistem terstruktur secara jelek yang sebenarnya merupakan masalah desain akan dibiarkan karena terkalahkan oleh trik implementasi.

Masalah pendekatan waterfall adalah ketidakluwesannya pembagian project ke dalam langkah yang nyata/jelas. Sistem yang disampaikan kadang-kadang tidak dapat

digunakan sesuai keinginan pelanggan. Namun demikian model waterfall mencerminkan kepraktisan engineering. Konsekuensinya, model proses perangkat lunak yang berdasarkan pada pendekatan ini digunakan dalam pengembangan sistem perangkat lunak dan hardware yang luas.

Pengembangan Evolusioner

Model ini berdasarkan pada ide pengembangan pada implementasi awal yang akan menghasilkan komentar pemakai sehingga dapat dilakukan perbaikan melalui banyak versi sampai sistem yang mencukupi dapat dikembangkan. Selain memiliki aktivitas-aktivitas yang terpisah model ini memberikan feedback dengan cepat dan serentak

Terdapat 2 tipe pada model ini

1. Pemrograman evolusioner

Dimana tujuan proses adalah bekerjasama dengan pelanggan untuk menghasilkan kebutuhan-kebutuhan dan menyampaikan sistem akhir kepada pemakai/pelanggan. Pengembangan dimulai dengan bagian-bagian sistem yang dimengerti. Sistem dikembangkan melalui penambahan features sesuai yang diusulkan oleh pelanggan.

2. Pemodelan

Dimana tujuan pengembangan evolusioner pada tipe ini adalah mengetahui kebutuhan-kebutuhan pelanggan dan mengembangkan definisi kebutuhan yang lebih baik untuk sistem. Model/contoh difokuskan pada penelitian bagian-bagian kebutuhan pelanggan yang kurang dimengerti.

Pemrograman evolusioner penting saat sulit untuk membuat spesifikasi sistem secara rinci. Beberapa orang mungkin setuju bahwa semua sistem masuk dalam tipe ini. Namun, pemrograman evolusioner banyak digunakan dalam pengembangan sistem *AI (artificial intelligence)* yang berusaha untuk menyamai kemampuan manusia.

Kita tidak mungkin membuat spesifikasi yang rinci untuk perangkat lunak yang menyamai manusia karena kita tidak mengerti bagaimana manusia menjalankan tugas-tugas mereka. Pendekatan evolusioner biasanya lebih efektif daripada pendekatan waterfall untuk hal pengembangan perangkat lunak yang harus dengan segera dapat memenuhi kebutuhan pelanggan. Namun, dari segi teknik dan manajemen, model ini memiliki masalah mendasar yaitu:

- Proses tidak visibel.

Manager-manager membutuhkan "deliverables" yang teratur untuk mengukur kemajuan. Jika sistem dikembangkan dengan cepat akan terjadi pemborosan pada pembuatan dokumen yang menggambarkan setiap versi sistem.

- Sistem-sistem biasanya kurang terstruktur

Kecenderungan perubahan yang terus menerus akan mengurangi struktur dari perangkat lunak. Evolusi perangkat lunak terlihat sulit dan mahal.

- Ketrampilan khusus jarang dimiliki

Tidak jelas batasan ketrampilan yang normal dalam rekayasa perangkat lunak yang mungkin dapat digunakan secara efektif dalam model pengembangan ini. Kebanyakan sistem yang dikembangkan melalui cara ini telah diimplementasikan oleh kelompok kecil yang memiliki ketrampilan yang tinggi dan motivasi yang kuat.

Untuk memecahkan masalah-masalah tersebut, kadang-kadang tujuan dari pengembangan evolusioner adalah mengembangkan contoh sistem. Contoh ini digunakan untuk mengerti dan mevalidasi spesifikasi sistem. Disinilah pengembangan evolusioner merupakan bagian dari beberapa proses yang lebih luas. (seperti model waterfall).

Karena masalah-masalah tersebut, sistem dengan skala besar biasanya tidak dikembangkan melalui cara ini. Pengembangan evolusioner lebih tepat untuk Pengembangan sistem yang relatif kecil.

Masalah-masalah mengenai perubahan sistem yang ada dihindari dengan meimplementasi ulang sistem keseluruhan kapanpun perubahan yang signifikan diperlukan. Jika pemodelan digunakan, tidak terlalu mahal.

Pengembangan sistem yang memiliki masa hidup yang relatif singkat.

Disini, sistem dikembangkan untuk mendukung beberapa aktivitas yang dibatasi oleh waktu. Contohnya, sebuah sistem yang mungkin dikembangkan secara khusus untuk peluncuran produk baru.

Pengembangan sistem atau bagian-bagian dari sistem yang besar dimana tidak memungkinkan untuk menyatakan spesifikasi secara rinci. Contohnya, sistem AI dan interfaces pemakai.

Spiral Boehm

Model proses nyata waterfall yang berorientasi dokumen telah diambil sebagai standar umum oleh banyak agen pemerintah dan pembuat perangkat lunak. Jadi, tidak mudah melupakan model tersebut walaupun masih terdapat masalah-masalah yang ditimbulkan dalam model tersebut. Kita membutuhkan sebuah proses yang lebih baik

untuk manajemen yang dapat menggunakan semua model umum seperti yang telah kita bicarakan sebelumnya. Model perbaikan tersebut juga harus memenuhi kebutuhan-kebutuhan pembuat perangkat lunak. Pendekatan alternatif diusulkan oleh Boehm (1988). Boehm mengusulkan sebuah model yang secara eksplisit menjelaskan bahwa resiko yang disadari mungkin membentuk dasar model proses umum.

Model Boehm berbentuk spiral. Setiap loop mewakili sebuah tahap dari proses perangkat lunak.

Tidak ada tahap yang tetap dalam model ini. Manajemen harus memutuskan bagaimana membentuk proyek kedalam tahap-tahap. Perusahaan biasanya bekerja dengan beberapa model umum dengan tahap tambahan untuk proyek khusus atau ketika masalah-masalah ditemukan selama pembuatan proyek.

Setiap loop dibagi dalam 4 sektor

1. Pembuatan tujuan

Tujuan, hambatan dalam proses ataupun produk serta resiko-resiko proyek ditentukan. Rencana rinci manajemen juga ditulis lengkap. Pembuatan strategi-strategi alternatif direncanakan sesuai dengan resiko yang ada.

2. Perkiraan dan pengurangan resiko

Untuk setiap resiko yang telah diidentifikasi, akan dibuat analisis rincinya. Kemudian diambil langkah-langkah untuk mengurangi resiko. contohnya, jika ada resiko bahwa persyaratan-persyaratan tidak tepat maka sebuah model contoh mungkin dapat dikembangkan.

3. Pengembangan dan validasi

Setelah evaluasi resiko, sebuah model pengembangan untuk sistem dipilih. Misalnya, jika resiko interface pengguna yang dominan maka model pengembangan yang tepat mungkin pengembangan evolusioner dengan menggunakan model contoh (prototipe)

Jika resiko keselamatan yang diutamakan, model pengembangan yang sesuai adalah transformasi formal dan seterusnya. Model waterfall mungkin tepat digunakan jika resiko yang diutamakan adalah integrasi sistem.

4. Perencanaan

Jika diputuskan untuk melanjutkan pada loop spiral berikutnya maka proyek dibicarakan kembali dan rencana dibuat untuk tahap selanjutnya.

Tidak perlu untuk menggunakan satu model tunggal pada setiap loop spiral bahkan dalam keseluruhan sisten perangkat lunak. Model spiral encompasses model lainnya. Pemodelan digunakan pada salah satu psiral untuk memecahkan masalah

kebutuhan. Kemudian dapat diikuti oleh model konvensional, waterfall. Transformasi formal digunakan untuk mengembangkan bagian-bagian sistem yang memiliki persyaratan keselamatan yang tinggi dan pendekatan reuse digunakan untuk pengimplementasian bagian-bagian lain dari sistem data manajemen.

Pada implementasinya, model spiral ini juga banyak digunakan, tetapi biasanya dikombinasikan dengan model yang lain. Pemodelan waterfall, yang sangat bagus dalam menentukan milestones dan pemodelan spiral, yang sangat bagus dengan menggunakan prototyping, merupakan kombinasi yang sering dipakai di dalam kontrak-kontrak untuk perangkat lunak dewasa ini.

Manajemen Resiko

Perbedaan yang mendasar antara model spiral dengan model lainnya adalah bahwa model spiral dengan eksplisit menyadari resiko-resiko yang ada. Resiko adalah konsep yang sulit didefinisikan secara tepat. Secara informal resiko adalah sesuatu yang sederhana yang dapat menyebabkan kesalahan. Contohnya, jika bertujuan menggunakan pemrograman bahasa baru (new programming language), resiko yang mungkin adalah alat pengumpul yang digunakan tidak reliabel dan tidak menghasilkan code objek yang efisien.

Resiko adalah sebagai hasil ketidakcukupan informasi. Resiko tersebut dapat dipecahkan dengan pengenalan beberapa kegiatan yang dapat menutupi informasi yang kurang menyakinkan. Dalam contoh diatas, resiko mungkin dapat diatasi dengan survey pasar untuk menemukan alat pengumpul mana yang dapat digunakan dan bagaimana kebaikan alat tersebut. Jika sistem ternyata tidak sesuai maka keputusan untuk menggunakan bahasa baru harus diubah.

Siklus spiral dimulai dengan penguraian tujuan-tujuan seperti performance, kegunaan, dan seterusnya. Cara alternatif dalam pencapaian tujuan dan hambatan dipergunakan dengan sebaik-baiknya kemudian diperhitungkan. Setiap alternatif diperhitungkan bertentangan dengan tujuan. Ini biasanya menghasilkan identifikasi sumber resiko proyek. Langkah selanjutnya adalah mengevaluasi resiko-resiko ini dengan aktivitas seperti analisis yang lebih detail, pembuatan model/contoh, simulasi dan seterusnya. Untuk menggunakan model spiral, Boehm menyarankan sebuah bentuk umum yang dipenuhi dalam setiap daerah spiral. Bentuk ini mungkin dilengkapi pada sebuah level abstrak atau perkiraan rinci yang imbang dari pengembangan produk.

1.1 PEMODELAN DATA

Pemodelan data menjawab serangkaian pertanyaan spesifik yang relevan dengan aplikasi pemrosesan data. Apakah objek data utama yang akan diproses oleh system ? Bagaimana komposisi dari masing-masing objek data dan atribut apa yang menggambarkan objek tersebut? Dimana objek saat ini berada? Bagaimana hubungan antara masing-masing objek data dan objek yang lainnya? Bagaimana hubungan objek dengan proses yang mentransformasikannya?

Untuk menjawab pertanyaan-pertanyaan tersebut, metode pemodelan data menggunakan **ERD**. ERD hanya berfokus pada data (sehingga memuaskan prinsip pertama analisis operasional).

2.1 Objek Data, Atribut Dan Hubungan

Model data terdiri dari tiga informasi yang saling bergantung :

1. **Objek Data** adalah representasi dari hampir semua informasi gabungan yang harus dipahami oleh perangkat lunak. Maksudnya dengan informasi gabungan kita mengartikan sesuatu yang memiliki sejumlah sifat atau atribut yang berbeda. Contohnya orang atau mobil dapat dipandang sebagai objek data bila salah satu dari mereka dapat didefinisikan dalam bentuk atribut.

2. **Atribut** menentukan properti suatu objek data dan mengambil salah satu dari tiga karakter hyang berbeda. Atribut dapat digunakan untuk :

1. **Menamai sebuah contoh dari objek data**

2. **Menggambar Contoh**

3. **Membuat referensi kecontoh ke contoh yang lain pada tabel yang lain.**

Sebagai tambahan, satu atribut atau lebih harus didefinisikan sebagai sebuah pengidentifikasi dimana atribut pengidentifikasi akan menjadi sebuah “kunci”. Dalam banyak kasus harga untuk mengidentifikasi adalah unik, meskipun hal itu bukan merupakan persyaratan. Dengan mengacu pada objek data mobil, pengidentifikasi yang bertanggung jawab dapat menjadi ID #.

3. **Hubungan** objek data disambungkan satu dengan yang lainnya dengan berbagai macam cara. Andaikan ada dua objek data BUKU dan TOKO BUKU, objek tersebut dapat diwakilkan dengan menggunakan notasi sederhana . misalnya :

- Toko buku memesan buku
- Toko buku menampilkan buku
- Took buku menstok buku

- Toko buku menjual buku
- Toko buku mengembalikan buku

Penting untuk dicatat bahwa objek relationship pairs mempunyai dua arah, dimana mereka dapat dibaca dari dua arah. Toko buku memesan buku atau buku dipesan oleh toko buku.

2.2 Kardinalitas dan Modalitas

Kardinalitas Model data harus dapat merepresentasikan jumlah peristiwa dari objek didalam hubungan yang diberikan. *Tiilmann (TIL. 93)* mendefinisikan kardinalitas dari objek – relationship pair dengan cara sebagai berikut :

Kardinalitas merupakan spesifikasi dari sejumlah peristiwa dari suatu (objek) yang dapat dihubungkan kesejumlah peristiwa dari (objek) yang lain. Kardinalitas biasanya diexpresikan secara sederhana ‘satu’ atau ‘banyak’. Dengan mempertimbangkan semua kombinasi dari ‘satu’ dan ‘banyak’ dua objek dapat dihubungkan sebagai :

- Satu ke satu (1:1) \longrightarrow suatu peristiwa dari objek A dapat berhubungan dengan satu dan hanya kejadian dari objek B, dan sebuah peristiwa dari B hanya dapat berhubungan dari satu kejadian A, misalnya : seorang suami hanya dapat memiliki satu orang istri dan seorang istri hanya dapat memiliki satu orang suami (di New Jersey).
- Satu ke banyak (1:N) \longrightarrow suatu kejadian A dapat berhubungan dengan satu atau lebih kejadian dari objek B, tetapi sebuah kejadian B dapat berhubungan dengan satu kejadian A, misalnya : seorang ibu dapat memiliki banyak anak, tetapi seorang anak hanya dapat memiliki satu orang ibu saja.
- Banyak ke banyak (N:N) \longrightarrow sebuah kejadian A dapat berhubungan dengan satu atau lebih kejadian dari B, sementara itu sebuah kejadian dari B dapat berhubungan dengan satu atau lebih kejadian dari A, misalnya : seorang paman dapat memiliki banyak keponakan sementara itu seorang keponakan dapat memiliki banyak paman.

Modalitas dari suatu hubungan adalah nol bila tidak ada kebutuhan eksplisit untuk hubungan yang terjadi atau hubungan itu bersifat optional. modalitas bernilai satu apabila suatu kejadian dari hubungan merupakan perintah.

2.3 Entity – Relationship Diagram

Objec-Relationship Pair merupakan batu pertama dari model data. Pasangan ini dapat diwakili secara grafis dengan menggunakan ERD. ERD pada mulanya diusulkan oleh **Peter Chen** (CHE77) untuk desain system database relasional dan telah dikembangkan. Tujuan utama dari ERD adalah untuk mewakili objek data dan hubungan mereka.

Objek data diwakili oleh sebuah persegi panjang yang diberi label. Hubungan ditunjukkan dengan garis yang diberi label yang menghubungkan objek dalam variasi ERD, garis yang menghubungkan berisi sebuah label permata yang diberi label dengan hubungan tersebut. Sambungan antara data dan objek dan hubungan dibangun dengan menggunakan berbagai macam simbol khusus yang menunjukkan kardinalitas dan modalitas.

Pemodelan data dan ERD memberi notasi yang singkat untuk mengamati data didalam konteks aplikasi pemrosesan data kepada analis. Dalam sebbagian besar kasus, pendekatan pemodelan data digunakan untuk menciptakan satu potong analisis, tetapi dia juga dapat digunakan untuk perancangan database dan untuk mendukung metode analisis persyaratan yang lain.

1.2 PEMODELAN FUNGSIONAL DAN ALIRAN INFORMASI

Analisis terstruktur dimulai sebagai sebuah teknik pemodelan aliran informasi. Sebuah sistem berbasis komputer direpresentasikan sebagai sebuah transformasi informasi. Sebagai contoh terlihat pada gambar 4.0 keseluruhan fungsi dari sistem tersebut diwakilkan sebagai transformasi informasi tunggal, yang ditulis sebagai gelembung didalam gambar. Satu input atau lebih diperlihatkan oleh anak panah yang diberi label , berasal dari entitas eksternal. Yang direpresentasikan sebagai sebuah kotak. Input mengendalikan transformasi tersebut untuk memproduksi informasi Output yang dilewatkan ke entitas eksternal.

2.1 Diagram Aliran Data

Pada saat informasi mengalir melalui pernakat lunak, dia dia dimodifikasi oleh suatu deretan transformasi. *Diagram aliran data / data flow diagram (DFD)* adalah sebuah teknik grafis yang menggambarkan aliran informasi dan transformasi yang diaplikasikan pada saat data bergerak dari input menjadi output. Bentuk dasar dari suatu aliran data. DFD juga dikenali sebagai grafik aliran aliran data atau *bubble chart*.

DFD tingkat 0 yang disebut juga dengan *model system fundamentasi* atau model konteks, merepresentasi seluruh elemen system sebagai sebuah bubble tunggal dengan data input dan output yang ditunjukkan oleh anak panah yang masuk dan keluar secara berurutan. Proses tambahan (bubble) dan jalur aliran informasi direpresentasikan pada saat DFD tingkat 0 dipartisi untuk megungkap detail yang lebih. Contohnya, sebuah DFD tingkat 1 dapat berisi lima atau enam bubble dengan anak panah yang saling menghubungkan. Notasi dasar yang digunakan untuk menciptakan suatu DFD.

2.3 Ekstensi Ward dan Mellor

Ward dan Mellor memperluas notasi analisis struktur dasar untuk mengakomodasi permintaan yang dikenakan oleh system real-time berikut ini :

- Aliran informasi dikumpulkan atau dihasilkan pada basis time-continious
- Informasi control yang dilewatkan melalui system dan pemrosesan control yang sesuai.
- Contoh bertingkat dari transformasi yang sama, yang kadang-kadang terjadi didalam situasi multitasking
- Pernyataan system dan mekanisme yang menyebabkan transisi diantara keadaan.

Dalam presentasi yang berarti dari aplikasi real-time, system harus memonitor informasi *time-cintinuous* yang digenerasikan oleh proses dunia nyata. Notasi aliran data konvensional tidak membuat perbedaan diantara data diskrit dan data *time-continuous* ekstensi untuk analisis terstruktur.

2.4 Ekstensi Hatley dan Pirbhai

Ekstensi Hatlei dan Pirbhai kenotasi analisis terstruktur dasar kurang berfokus pada kreasi dari symbol grafis tambahan dan lebih berfokus pada representasi dan spesifikasi aspek perangkat lunak yang berorientasi pada control.

1.3 PEMODELAN TINGKAH LAKU

Pemodelan tingkah laku merupakan suatu prinsip operasional untuk semua metode analisis persyaratan tetapi hanya versi analisis terstruktur yang luas yang memberikan suatu notasi bagi tipe pemodelan ini. Untuk menggambarkan penggunaan

ekstensi control dan tingkah laku Hatley dan Pirbhai, diandaikan perangkat lunak embedded dalam sebuah mesin foto kopi. Foto kopi tersebut melakukan sejumlah fungsi yang diimplikasikan oleh DFD tingkat 1. perlu dicatat bahwa penyaringan tambahan dari aliran dan definisi dari masing-masing item akan diperlukan.

1.4 MEKANIK DARI ANALISIS TERSTRUKTUR

2.1 Membuat sebuah diagram hubungan Entitas

Diagram hubungan entitas memungkinkan seorang perekayasa perangkat lunak untuk secara penuh menspesifikasikan objek data yang merupakan input dan output dari system. Pendekatan berikut ini perlu diketahui dalam membuat diagram Entitas :

- Selama pengumpulan persyaratan, pelanggan diminta untuk mendaftar 'hal-hal' yang akan dituju oleh proses bisnis dan aplikasi. 'Hal-hal' ini dimasukkan kedalam sebuah daftar objek data input dan output dan entitas eksternal yang menghasilkan atau mengkonsumsi informasi.
- Dengan mengambil objek satu pada satu saat , analis dan pelanggan mendefinisikan apakah ada sambungan (tidak diberi nama pada tahap ini) ada diantara objek data dan objek lain.
- Dimanapun sambungan ada, analis dan pelanggan menciptakan satu pasangan hubungan objek atau lebih .
- Untuk masing-masing pasangan hubungan objek, dicari kardinalitas dan modalitas.
- Langkah 2 sampai 4 dilanjutkan secara iterative sampai semua pasangan hubungan objek sudah didefinisikan. Sudah menjadi kebiasaan untuk menemukan penghilangan pada saat proses ini berlanjut. Objek dan hubungan baru akan ditambahkan pada saat jumlah iterasi bertambah.
- Atribut dari masing-masing entitas didefinisikan
- Diagram entitas diformalisasikan dan dikaji
- Langkah 1 sampai 7 diulangi sampai pemodelan data terlengkapi.
-

2.2 Membuat Sebuah Model Aliran Data

Diagram aliran data (DFD) memungkinkan perekayasa perangkat lunak untuk mengembangkan model domain informasi dan domain fungsional pada saat yang sama. Beberapa tuntunan sederhana dengan terukur dapat membantu selama derivasi sebuah diagram aliran data :

1. diagram aliran data tingkat 0 harus menggambarkan perangkat lunak/system sebagai gelembung tunggal.
2. input dan output utama harus dicatat secara berhati – hati
3. penyaringan harus dimulai dengan mengisolasi proses calon, objek data, dan penyimpanan yang akan direpresentasikan pada tingkat selanjutnya.
4. semua anak panah dan gelembung harus diberi label dengan nama yang berarti
5. *kontinuitas aliran informasi* harus dijaga dari tingkat ke tingkat
6. satu gelembung pada satu saat harus disaring.

Ada kecenderungan natural untuk terlalu mengkomplikasikan diagram aliran data. Hal ini terjadi bila analisis ingin menunjukkan terlalu banyak detail pada saat yang terlalu dini

2.3 Membuat Sebuah Model Aliran Kontrol

Untuk beberapa tipe aplikasi pemrosesan, model data dan diagram aliran data merupakan hal yang diperlukan untuk memperoleh wawasan yang berarti kedalam persyaratan perangkat lunak. Tetapi, seperti yang telah dicatat, disana ada suatu kelas aplikasi yang besar yang lebih dikendalikan oleh kejadian dari pada data, yang lebih menghasilkan informasi control dari pada menghasilkan laporan dan tampilan. Dan yang memproses informasi dengan perhatian besar kepada waktu dan kinerja kerja. Aplikasi semacam itu mambutuhkan pemodelan aliran control sebagai tambahan kepemodelan aliran data.

Telah kita catat bahwa sebuah kejadian atau item control diimplementasikan sebagai harga Boolean (misalnya; benar atau salah, on atau off, 1 atau 0) atau sebuah daftar diskrit dari keadaan (kosong,penuh), untuk memilih calon kejadian yang potensial, diusulkan tuntutan berikut ini :

- Daftarlh semua sensor yang dibaca oleh perangkat lunak
- Daftarlh semua keadaan interupsi
- Bacalah semua saklar yang diaktuasi oleh operator
- Daftarlh semua keadaan data
- Dengan menarik uraian data kerja dan data benda yang diaplikasikan ke narasi pemrosesan, kajilah semua item control sebagai input /output CSPEC yang mungkin

- Gambarkanlah tingkah laku dari system dengan mengidentifikasi keadaannya ; identifikasikanlah bagaimana keadaan dicapai dan definisikanlah transisi antar keadaan.
- Fokuskanlah penghilangan yang mungkin sebuah kesalahan yang paling umum didalam menspesifikasikan control (misalnya, tanyakanlah ; adakah suatu cara dimana saya dapat masuk ke keadaan itu atau keluar darinya).

2.4 Spesifikasi Kontrol

CSPEC mempresentasikan tingkah laku system (pada tingkat dimana dia direferensikan) didalam dua cara yang berbeda. CSPEC berisi sebuah diagram transisi keadaan (STD) yang merupakan suatu *spesifikasi sekuensial* dari tingkah laku. Dia juga dapat berisi suatu table aktifitas proses (PAT) – sebuah spesifikasi kombinatorial dari tingkah laku.

2.5 Spesifikasi Proses

Spesifikasi Proses (PSPEC) digunakan untuk menggambarkan semua proses model aliran yang nampak pada tingkat akhir penyaringan. Kandungan dari spesifikasi proses dapat termasuk teks naratif, bahasa design program/*Programme Design Language* (PDL) dari Algoritma proses, persamaan Matematika, table, diagram atau bagan, dengan memberikan sebuah PSPEC untuk mengiringi masing-masing gelembung didalam model aliran, berarti perekrayasa perangkat lunak menciptakan sebuah “spesifikasi mini” yang dapat berfungsi sebagai sebuah langkah pertama didalam kreasi spesifikasi persyaratan perangkat lunak dan sebagai penuntun bagi desain komponen program yang akan mengimplementasikan program.

PSPEC : Naratif Pemrosesan untuk segi tiga Analisis

Proses segitiga analisis menerima nilai A,B dan C yang menyajikan dimensi sisi sebuah segitiga. Proses memeriksa nilai-nilai dimensi untuk menentukan apakah semua nilai positif, jika ditemukan nilai negative, akan muncul pesan error. Proses mengevaluasi keabsahandata input untuk menentukan apakah dimensi menentukan. Keabsahan segitiga, dan jika ya, apa tipe segitiga sama sisi, sama kaki , atau tidak sama sisi yang diimplikasikan oleh dimensi tipe adalah output.

Spesifikasi Proses untuk Proses PDF

PSPEC : Naratif Pemrosesan untuk segi tiga Analisis

Prosedur Analisa Segitiga :

Membaca dimensi sisi-sisi segitiga;

Jika semua dimensi negatif maka terjadi pesan error

Jika dimensi terbesar kurang dari jumlah yang lain maka mulai

Tentukan jumlah sama sisi

Jika tiga sisi sama maka tipenya adalah sama sisi ;

Jika dua sisi sama maka tipenya adalah sama kaki

Jika tidak ada sisi yang sama maka tipenya adalah tidak sama output tipe segitiga

End

Tipe output lain = 0 indikasi bahwa tidak ada segitiga;

Endif

Enproc

Spesifikasi Proses Menggunakan PDL untuk proses DFD

1.5 KAMUS DATA

Kamus data telah diusulkan sebagai sebuah tata bahasa quasi-formal untuk menggambarkan kandungan dari objek yang didefinisikan selama analisis terstruktur. Notasi pemodelan yang penting ini telah didefinisikan sebagai berikut : Kamus data merupakan sebuah daftar yang terorganisasi dari elemen data yang berhubungan dengan system, dengan definisi yang tegas dan teliti sehingga pemakai dan analisis system akan memiliki pemahaman yang umum mengenai input, output, komponen penyimpanan , dan bahkan kalkulasi inter-mediate.

Saat ini, kamus data hamper selalu diimplementasikan sebagai bagian dari sebuah “piranti desain dan analisis terstruktur “ CASE. Sebagian kamus data berisi informasi sebagai berikut :

- Name = sebenarnya dari data atau item control, penyimpanan data, atau entitas eksternal.
- Aliasi = nama lain yang digunakan untuk entri pertama
- Where-used/how used = suatu daftar dari proses yang menggunakan data atau item control dan bagaimana dia digunakan (misalnya input ke progress, output dari progress, sebagai suatu penyimpanan, sebagai suatu entitas eksternal)
- Content description = suatu notasi untuk mempresentasikan isi
- Supplementary information = informasi lain mengenai tipe data, harga preset (bila diketahui).

Notasi yang digunakan untuk mengembangkan diskripsi isi, yang diilustrasikan didalam Gambar 9.0 memungkinkan analisis untuk mempresentasikan data komposit (misal objek data) didalam salah satu dari tiga fundamenta yang dapat dikonstruksi olehnya :

1. sebagai sebuah urutan item data
2. sebagai suatu pilihan dari antara serangkaian item data atau
3. sebagai sebuah kelompok pengulangan item data

Konstruksi data	Notasi	Arti
Berurutan	=	Disusun atas
Pilihan	+	Dan
Pengulangan	[]	Baik ini ,atau
	{ } ⁿ	Pengulangan ke-n dari
	{ }	Data opsional
	. .	Komentar tidak dibatasi

Masing-masing entri item data direpresentasikan sebagai bagian dari urutan, seleksi dan pengulangan dapat menjadi objek data lain yang memerlukan penyaringan lebih jauh lagi didalam kamus.

1.6 OVERVIEW MENGENAI METODE ANALISIS KLASIK

Data Structured Systems Development

Data Structure System Development (DSSD), yang disebut juga dengan metodologi Warnier-Orr terjadi dari kerja perintis mengenai analisis domain informasi yang dilakukan oleh J.D Warnier. Warnier mengembangkan sebuah notasi untuk mempresentasikan hirarki informasi dengan menggunakan tiga konstruksi untuk urutan, pemilihan, dan pengulangan dan mendemonstrasikan bahwa struktur perangkat lunak dapat ditarik dari struktur data.

Ken Orr memperluas kerja Warnier untuk mencakup pandangan yang lebih luas mengenai domain informasi yang telah dikembangkan kedalam DSSD

Jackson System Development

Jackson System Development (JDS) mengembangkan kerja yang dilakukan oleh M.A. Jackson tentang analisis domain informasi dan hubungannya dengan desain system dan program. Dalam kalimat Jackson , “Pengembang memulai

dengan menciptakan sebuah model realistis dimana system diperhatikan, realitas yang memperlengkapi masalah subjek (system)nya..”

SADT

Structured analysis and design technique (SADT) adalah sebuah teknik yang telah digunakan secara luas sebagai sebuah notasi untuk definisi system, representasi proses, analisis persyaratan perangkat lunak dan desain system /perangkat lunak.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman
- Roger S. Pressman, Ph.D, *“Rekayasa Perangkat Lunak”*, Pendekatan Praktisi (Buku Satu), ANDI Yogyakarta
- Roger S. Pressman, *"Software Engineering, a Practitioner's Approach"* Fourth Edition, McGraw Hill, 1997.
- Barbee Teasley Mynatt, *"Software Engineering with Student Project Guidance"*, Prentice Hall Int. 1990.
- Roger S. Pressman, *"Software Engineering, A Beginner's Guide"*, McGraw Hill, 1998.



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Desain perangkat lunak dan rekayasa perangkat lunak, prinsip desain dan konsep desain

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka
10

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Materi ini membahas mengenai pembuatan desain perangkat lunak dan merekayasakannya, Mengenal prinsip desain beserta konsepnya

Kompetensi

Mahasiswa dapat mendisain perangkat lunak dengan cara merekayasakan disesuaikan dengan konsep

Pada bab ini perancangan desain yang akan dibahas merupakan perancangan terstruktur lanjutan tahapan analisa terstruktur. Perancangan perangkat lunak merupakan inti teknik dari proses rekayasa perangkat lunak dan merupakan aktivitas pertama dari tiga aktivitas teknik – perancangan, pembuatan kode, dan pengujian – yang diperlukan untuk membangun dan menguji perangkat lunak.

Pengertian

Perancangan perangkat lunak dapat didefinisikan sebagai

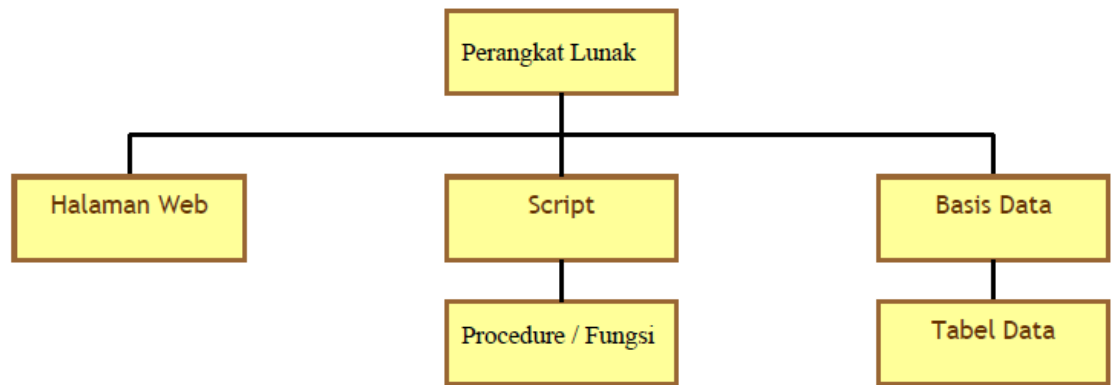
- Proses untuk mendefinisikan suatu model atau rancangan perangkat lunak dengan menggunakan teknik dan prinsip tertentu sedemikian hingga model atau rancangan tersebut dapat diwujudkan menjadi perangkat lunak.
- Proses mendefinisikan arsitektur perangkat lunak, komponen, modul, antarmuka, pendekatan pengujian, serta data untuk memenuhi kebutuhan yang sudah ditentukan sebelumnya. [IEE98]
- Proses bertahap dimana semua kebutuhan yang ada diterjemahkan menjadi suatu cetak biru yang akan digunakan untuk mengkonstruksi perangkat lunak. [PRE01]

Tujuan dilakukannya perancangan oleh seorang designer system (software engineer) adalah

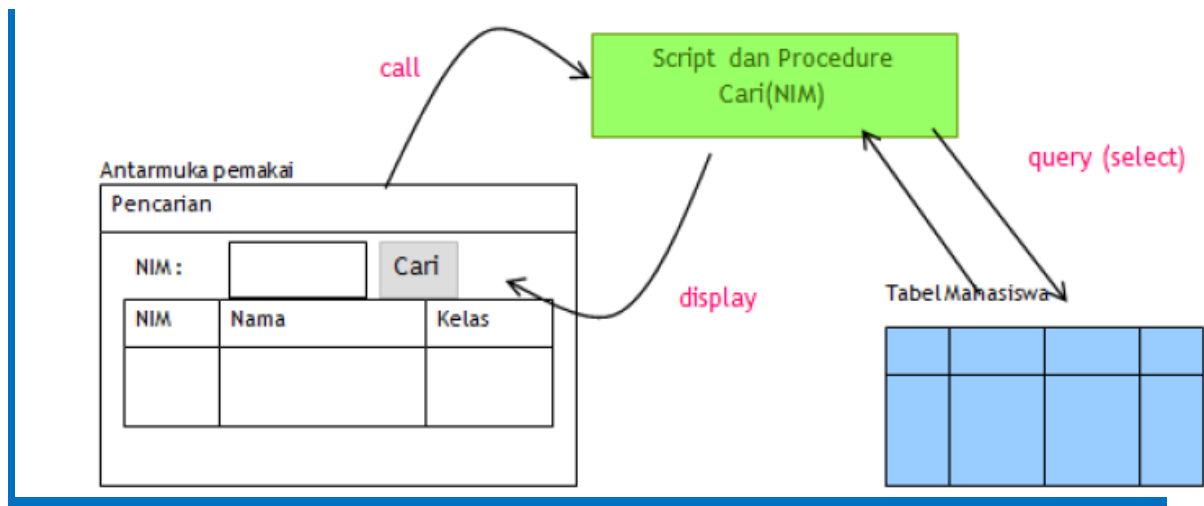
- Mendekomposisi sistem (perangkat lunak) menjadi komponen-komponennya (data, antarmuka, prosedur, arsitektur). Sebagai gambaran, pada gambar 5.1 menunjukkan dekomposisi perangkat lunak menjadi halaman web (antarmuka), script (prosedur) dan basisdata/tabel data (desain data).
- Menentukan relasi antar komponen.
- Menentukan mekanisme komunikasi antar komponen. Sebagai gambaran, pada gambar 5.2 yang menunjukkan mekanisme dan relasi antar komponen perangkat lunak yaitu relasi antarmuka pemakai ke prosedur/script untuk meminta sebuah data yang diinginkan pengguna serta bagaimana sebuah

prosedur mengakses tabel data agar dapat ditampilkan sesuai dengan permintaan pengguna pada antarmuka pengguna.

- Menentukan antarmuka komponen.
- Menjelaskan fungsionalitas masing-masing komponen.



Gambar 1 Dekomposisi perangkat lunak menjadi komponen-komponennya.



Gambar 2 Mekanisme dan relasi antar komponen perangkat lunak.

Prinsip Perancangan

Perancangan perangkat lunak merupakan model dan proses. Proses perancangan merupakan serangkaian langkah yang memungkinkan seorang desainer menggambarkan semua aspek perangkat lunak yang dibangun, sedangkan model perancangan hampir sama dengan rencana arsitek untuk sebuah rumah yaitu memulai dengan menyajikan totalitas hal yang akan dibangun (misal pandangan 3 dimensi dari

rumah yang akan dibangun, setelah itu akan disaring hal-hal yang memberikan panduan bagi pembangunan setiap detail dari rumah, seperti layout ruangan, layout pipa dan lainnya). Sama halnya dengan model perancangan yang dibuat untuk perangkat lunak memberikan berbagai pandangan yang berbeda terhadap program komputer.

Ada beberapa prinsip yang dikemukakan oleh Davis [DAV95] yang perlu diketahui oleh desainer untuk dapat mengendalikan proses perancangan, yaitu

- Perancangan harus dapat ditelusuri sampai ke model analisis.
- Perancangan tidak boleh berulang, maksudnya dapat menggunakan kembali rancangan yang sudah ada sebelumnya (reusable component).
- Perancangan dapat diperbaiki atau diubah tanpa merusak keseluruhan sistem.
- Perancangan harus dinilai kualitasnya pada saat perancangan, bukan setelah sistem jadi dengan kata lain siap diimplementasikan.
- Perancangan harus mempunyai beberapa pendekatan alternatif rancangan.
- Perancangan harus mengungkap keseragaman dan integrasi
- Perancangan harus meminimalkan kesenjangan intelektual antara perangkat lunak dan masalah yang ada di dunia nyata. Maksudnya perancangan perangkat lunak harus mencerminkan struktur domain permasalahan.
- Perancangan bukanlah pengkodean dan pengkodean bukanlah perancangan.
- Perancangan harus dikaji untuk meminimalkan kesalahan-kesalahan konseptual. Desainer harus menekankan pada hal-hal yang penting seperti elemen-elemen konseptual (ambiguitas, inkonsisten).

Jika prinsip perancangan diatas diaplikasikan dengan baik, maka desainer telah mampu menciptakan sebuah perancangan yang mengungkapkan faktor-faktor kualitas eksternal dan internal[MEY88]. Faktor-faktor eksternal adalah sifat-sifat perangkat lunak yang dapat diamati oleh pemakai (misal kecepatan, reliabilitas, ketepatan, usability). Sedangkan faktor internal lebih membawa pada perancangan berkualitas tinggi dan perspektif teknis dari perangkat lunak yang sangat penting bagi para perekrayasa perangkat lunak. Untuk mencapai kualitas faktor internal, seorang desainer harus memahami konsep-konsep dari perancangan perangkat lunak.

Konsep Perancangan

Pada dasarnya konsep perancangan memberikan kerangka kerja atau pedoman untuk mendapatkan perangkat lunak yang bisa berjalan dengan baik. Ada beberapa konsep perancangan yang dikemukakan oleh Pressman [PRE01] dan perlu dipahami oleh seorang desainer agar mendapatkan perancangan yang berkualitas tinggi yaitu

1. Abstraksi

Abstraksi merupakan cara untuk mengatur kompleksitas sistem dengan menekankan karakteristik yang penting dan menyembunyikan detail dari implementasi. Tiga mekanisme dasar dari abstraksi yaitu :

- a. Abstraksi Prosedural, urutan instruksi yang mempunyai sebuah nama yang menggambarkan fungsi tertentu
- b. Abstraksi Data, kumpulan data yang mempunyai nama yang menggambarkan objek data.
- c. Abstraksi Control, mengimplikasikan sebuah mekanisme kontrol dari program.

2. Dekomposisi

Dekomposisi merupakan mekanisme untuk merepresentasikan detail-detail dari fungsionalitas. Dengan adanya dekomposisi membantu para desainer mengungkapkan detail tingkat rendah ketika perancangan sedang berjalan. Jadi dekomposisi membagi perancangan secara top-down/menyaring tingkat detail dari prosedural.

3. Modularitas

Mekanisme membagi perangkat lunak ke dalam elemen-elemen kecil dan dapat dipanggil secara terpisah, biasanya elemen ini sering disebut dengan modul.

Modularitas merupakan karakteristik penting dalam perancangan yang baik karena

- a. Menyediakan pemisahan fungsionalitas yang ada pada perangkat lunak.
- b. Memungkinkan pengembang mengurangi kompleksitas dari sistem.

- c. Meningkatkan skalabilitas, sehingga perangkat lunak dapat dikembangkan oleh banyak personal.

Modularitas perangkat lunak ditentukan oleh coupling dan cohesion:

- a. Coupling: derajat ketergantungan antar modul yang berinteraksi.
- b. Cohesion: derajat kekuatan fungsional dalam suatu modul

Modul yang baik harus mempunyai cohesion yang tinggi dan coupling yang rendah.

Faktor-faktor yang mempengaruhi coupling:

- a. Banyaknya data yang dilewatkan antar modul (passing parameter)
- b. Banyaknya kontrol data yang dilewatkan antar modul.
- c. Banyaknya data global yang digunakan bersama oleh beberapa modul.

4. Arsitektur Perangkat Lunak

Arsitektur perangkat lunak merupakan struktur hirarki dari komponen program (modul), cara bagaimana komponen tersebut berinteraksi dan struktur data yang digunakan oleh komponen.

5. Hirarki Kontrol

Hirarki kontrol disebut juga dengan struktur program, yang merepresentasikan organisasi (hirarki) komponen program (modul) serta mengimplikasikan suatu hirarki kontrol. Hirarki kontrol tidak mengimplikasikan aspek prosedural dari perangkat lunak, seperti urutan proses, kejadian/urutan keputusan, atau pengulangan operasi.

Hirarki kontrol juga merepresentasikan dua karakteristik yang berbeda dari arsitektur perangkat lunak yaitu visibilitas dan konektivitas. Visibilitas menunjukkan serangkaian komponen program yang dapat diminta dan dipakai sebagai data oleh komponen yang diberikan dan dilakukan secara tidak langsung. Sedangkan konektivitas mengindikasikan serangkaian komponen program yang diminta secara tidak langsung atau digunakan data oleh sebuah modul yang ditetapkan.

6. Partisi Struktural

Struktur program harus dipartisi secara horisontal maupun struktural. Partisi ini membagi cabang-cabang yang terpisah dari hirarki modul untuk menjadi sebuah fungsi program. Ada beberapa keuntungan yang didapat mempartisi arsitektur secara horisontal, yaitu

- a. menghasilkan perangkat lunak yang mudah diuji.
- b. menghasilkan penyebaran efek samping yang sedikit.
- c. menghasilkan perangkat lunak yang lebih mudah diperluas.
- d. menghasilkan perangkat lunak yang lebih mudah dipelihara.

Selain struktur program bisa dipartisi secara horisontal bisa juga dipartisi secara vertikal, dimana kontrol dan kerja dari arsitektur program didistribusikan secara top-down.

7. Struktur Data

Struktur data merepresentasikan hubungan logis antara elemen-elemen data. Selain itu struktur data juga menentukan organisasi, metode akses, tingkat asosiativitas dan alternatif pemrosesan untuk informasi.

8. Prosedur Perangkat Lunak

Prosedur perangkat lunak lebih berfokus pada detail-detail pemrosesan dari masing-masing modul. Prosedur harus memberikan spesifikasi yang teliti terhadap pemrosesan, mencakup event, keputusan, operasi, dan struktur data.

9. Penyembunyian Informasi

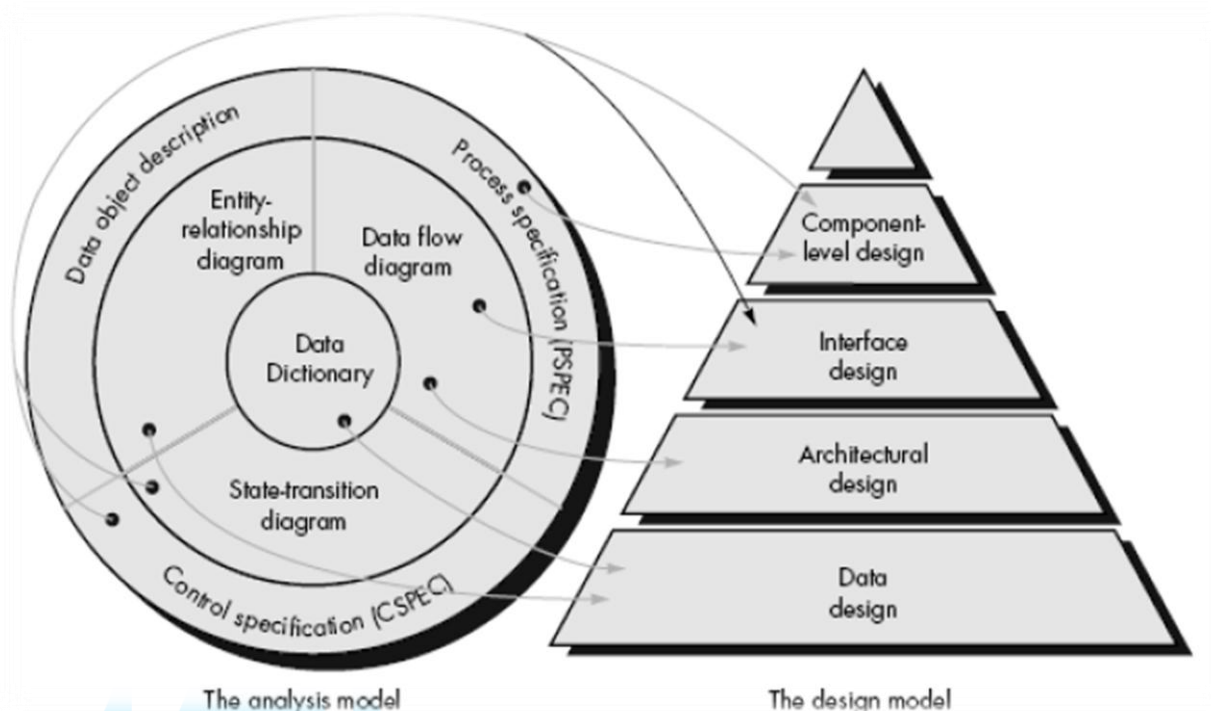
Sebuah mekanisme perancangan modul sehingga informasi yang terkandung dalam modul tidak dapat diakses oleh modul lain yang tidak berkepentingan dengan informasi tersebut. Informasi yang disembunyikan terdiri dari

- a. Representasi data
- b. Algoritma seperti teknik pengurutan dan pencarian

- c. Format masukan dan keluaran
- d. Perbedaan mekanisme/kebijakan
- e. Antarmuka modul tingkat rendah Ada beberapa alasan kenapa konsep perancangan ini perlu dipahami oleh desainer yaitu
 1. Mengatur sistem perangkat lunak yang kompleks.
 2. Meningkatkan kualitas faktor dari perangkat lunak.
 3. Memudahkan penggunaan kembali simantik sistem atau perangkat lunak.
 4. Memecahkan permasalahan-permasalahan perancangan yang ada pada umumnya.

Transformasi Model Analisa ke Perancangan

Masing-masing elemen pada model analisis (bab 4) akan memberikan informasi yang diperlukan untuk menciptakan model perancangan. Pada gambar 5.3 menunjukkan bagaimana menterjemahkan model analisis menjadi empat model perancangan.



Gambar 3 Transformasi model analisis ke model perancangan perangkat lunak

Pada tahap perancangan ini akan dihasilkan empat model/objek perancangan, yaitu

- Perancangan data, yang berupa tabel-tabel basis data / file data konvensional Dan struktur data internal (jika diperlukan).
- Perancangan arsitektur yang berupa Structure chart dan struktur menu program (sebagai pelengkap)
- Perancangan antarmuka (interface)
- Perancangan level komponen/prosedural yang berupa spesifikasi program (algoritma)

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman





MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Desain Modular efektif, model desain dan dokumen desain

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

11

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Materi ini membahas mengenai desain modular efektif, model desain dan dokumentasi desain.

Kompetensi

Mahasiswa dapat membuat berbagai desain untuk keperluan membuat Rekayasa Perangkat Lunak

Desain Software dan Software Engineering

Desain ->Langkah pertama dalam fase pengembangan untuk prodengineer manapun. Bertindak sebagai pondasi untuk semua software engineering dan pemeliharaan software dengan langkah-langkah yang mengikutinya.

Tujuan seorang perancang adalah menghasilkan suatu model(atau penyajian) dari sebuah entitas yang akan dibangun

Input desain software : Model analisa kebutuhan dan dokumentasi spesifikasi

Output desain software : Model desain dan dokumentasi spesifikasi desain

Desain - menerjemahkan kebutuhan ke model desain lengkap untuk sebuah produk software.

- menyediakan representasi software yang dapat diduga untuk kualitas.

Desain Software

Sejumlah metode desain dapat digunakan untuk menghasilkan desain software:

- Desain data: mentransformasi model domain informasi ke struktur data
- Desain arsitektur: menggambarkan relasi antar elemen struktural utama program
- Desain interface : mendeskripsikan bagaimana software berkomunikasi with user
- Desain prosedur: mentransformasikan elemen structural arsitektur program ke sebuah deskripsi prosedur dari komponen software.

Evolusi desain software:

- Konstruksi program modular dan metode perbaikan top-down
- Pemrograman terstruktur
- Perpindahan data flow/data structure ke sebuah definisi desain
- Pendekatan berorientasi object

Fitur umum metode desain software:

- Sebuah mekanisme untuk perpindahan sebuah model analisa ke representasi desain
- Sebuah notasi untuk merepresentasi komponen fungsional dan interface-nya.
- Heuristic untuk perbaikan dan partisi
- Panduan untuk assessment kualitas

Proses Desain

Desain software --> sebuah proses iteratif dimana kebutuhan diterjemahkan ke sebuah “blueprint” untuk mengkonstruksi software.

Desain direpresentasikan pada level atas abstraction. Saat iterasi desain terjadi, perbaikan subsequent membawa ke representasi desain pada level abstraksi yang lebih bawah.

Kualitas desain sangat penting.

Dua metode digunakan untuk mengecek kualitas:

- a) Tinjauan ulang teknis formal, dan
- b) desain walkthrough

Tiga fitur umum sebuah desain yang baik dari McGlaughlin’s [McG91]:

- Desain harus mengimplementasikan semua kebutuhan(eksplisit/implisit)
- Desain harus dapat dibaca dan dimengerti
- Desain harus menyediakan suatu gambar lengkap software pada aspek aspek data, fungsi dan perilaku.

Kualitas Desain

Untuk mengevaluasi sebuah desain software, sebuah criteris kualitas desain dapat digunakan.

Berikut panduan untuk sebuah desain yang baik:

- Sebuah desain harus A design should exhibit a hierarchical organization about software
- Sebuah desain harus modular berdasarkan pada partisi logical.

Sebuah desain terdiri dari data dan abstraksi prosedural.

Sebuah desain harus membawa ke modul dengan fitur fungsional yang independen.

Sebuah desain harus membawa interface yang sederhana antar modul.

Sebuah desain harus diturunkan menggunakan metode yang dapat berulang.

Proses desain software memacu desain yang baik dalam aplikasi dengan prinsip desain fundamental, metodologi yang sistematis, dan melalui review(pengulangan).

Prinsip Desain

David [DAV95] memaparkan sekumpulan prinsip untuk desain software:

- Proses desain seharusnya tidak bertahan dari “tunnel vision”.
- Desain harus dapat di trace ke model analisa.
- Desain seharusnya tidak menemukan/invent wheel.
- Desain harus “memperkecil jarak intellectual” antara software dan masalah-masalah pada dunia nyata.
- Desain harus mengeluarkan uniformity dan integrasi.
- Desain harus terstruktur untuk mengakomodasi perubahan.
- Desain harus terstruktur untuk mendegradasi secara halus(degrade gently).
- Desain bukan coding.
- Desain harus di-assessed untuk kualitas.

Desain harus diulang untuk meminimalis kesalahan konsep.

Faktor kualitas eksternal: diobservasi oleh user.

Faktor kualitas internal: penting untuk engineer.

Konsep Desain

Abstraksi:

Tiap langkah pada proses software engineering adalah sebuah perbaikan pada tingkatan abstraksi dari solusi software.

- Abstraksi data: Sekumpulan data

- Abstraksi prosedural:

Satu urutan instruksi pada sebuah fungsi spesifik

- Abstraksi kontrol:

Sebuah program mengontrol mekanisme tanpa menspesifikasikan detail internal.

Refinement(perbaikan): Perbaikan sebenarnya adalah sebuah proses dari elaborasi.

Stepwise dari perbaikan adalah sebuah strategi desain top-down yang dikeluarkan oleh Niklaus [WIR71].

Arsitektur sebuah program dikembangkan dengan sukses memperbaiki level detail procedural.

Proses perbaikan program analog terhadap proses perbaikan dan partisi yang digunakan selama analisa kebutuhan. Perbedaan utama berada pada level detail implementasi, disamping pendekatannya.

Abstraksi dan perbaikan secara komplemen dikonsep.

Abstraksi memungkinkan seorang desainer untuk menspesifikasi prosedur dan data w/o detail. Perbaikan membantu desainer untuk me-reveal detail low-level.

Konsep Desain – Modularity

Konsep modularity telah dipakai selama hampir empat dekade. Software dibagi komponen dengan nama dan alamat yang berbeda, disebut modul.

Meyer [MEY88] mendefenisikan lima kriteria yang memungkinkan kita mengevaluasi sebuah metode desain dengan bergantung kepada kemampuannya mendefenisikan sebuah sistem modular efektif:

Modular decomposability: sebuah metode desain menyediakan sebuah mekanisme sistematis untuk men-decompose atau membagi masalah ke sub-masalah mengurangi kompleksitas dan mendapatkan modularity

Modular composability: sebuah metode desain memungkinkan komponen desain yang telah ada dirakit ke sebuah sistem baru.

Modular understandability: sebuah modul dapat dimengerti sebagai sebuah unit yang berdiri sendiri dan akan lebih mudah membangun dan mengubahnya.

Modular continuity: perubahan kecil terhadap kebutuhan sistem menghasilkan perubahan pada tiap modul, dibanding perubahan system-wide.

Modular protection: sebuah kondisi aberrant terjadi dalam sebuah modul dan efeknya di-constrain dalam modul.

Arsitektur Software

Arsitektur software adalah struktur hirarki dari komponen program components dan interaksinya.

Shaw dan Garlan [SHA95a] menggambarkan sekumpulan properti desain arsitektur:

Properti structural : Desain arsitektur mendefenisikan komponen sistem dan interaksinya.

Properti extra-functional : Desain arsitektur harus meng-address bagaimana arsitektur desain menerima kebutuhan untuk performance, kapasitas, ketersediaan(reliability), adaptability, keamanan.

Kumpulan sistem yang berhubungan:

desain arsitektur harus menggambar pola yang dapat diulang pada desain dengan sistem yang sama.

Metode desain arsitektural yang berbeda:

Model structural: merepresentasikan arsitektur sebagai sebuah koleksi terorganisir dari komponen

Model framework: meningkatkan level desain abstraksi dengan mengidentifikasi secara berulang framework desain arsitektur(pola-pola)

Model dinamis: meng-address aspek-aspek perilaku arsitektur program

Model proses: fokus pada desain bisnis atau proses teknis

Model functional: dapat digunakan untuk merepresentasikan hierarki fungsional sebuah sistem

Partisi Structural

Struktur program harus dipartisi secara horizontal dan vertikal.

- (1) Partisi horizontal menggambarkan cabang-cabang yang terbagi dari hierarki modular untuk tiap fungsi program utama.

Cara termudah adalah mempartisi sebuah sistem menjadi:
input, transformasi data(pemrosesan), dan output

Keuntungan dari partisi horizontal:

- mudah untuk diuji, di-maintain, dan extend
- efek yang lebih kecil pada perubahan propagasi atau error propagasi

Kerugian: lebih banyak data dilewatkan melalui interface modul
--> menyulitkan kontrol keseluruhan dari aliran program

- (2) Partisi vertical memaparkan kontrol dan work harus terdistribusi top-down dalam struktur program.

Keuntungan: baik pada kesesuaian untuk perubahan:

- mudah untuk me-maintain perubahan
- mengurangi pengaruh perubahan dan propagasi.

Desain Modular Efektif

Informasi yang disimpan: Modul-modul seharusnya dispesifikasi dan didesain sehingga detail internal dari modul harus dapat terlihat atau dapat diakses terhadap modul lainnya.

Keuntungan utama: mengurangi pengaruh perubahan pada testing(pengujian) dan maintenance(perawatan).

Independen fungsional: Mendesain modul-modul berdasarkan fitur fungsional independen.

Keuntungan utama: modularity efektif

Cohesion: sebuah ekstensi alami dari konsep informasi yang disimpan sebuah modul dapat menampilkan sejumlah tugas

Sebuah modul cohesive menampilkan sebuah tugas tunggal dalam sebuah prosedur dengan interaksi kecil dengan lainnya.

Goal: untuk mencapai cohesion yang tinggi untuk modul-modul pada sebuah sistem.

Tipe-tipe yang berbeda dari cohesion:

- coincidentally cohesive: sekumpulan tugas yang berhubungan satu sama lain secara bebas.
- logically cohesive: koneksi logis antara elemen-elemen pemrosesan
- communication cohesion: data di-share antar elemen-elemen pemrosesan
- procedural cohesion: pemesanan antara elemen-elemen pemrosesan

Desain Modular Efektif

Suatu ukuran interkoneksi antar modul-modul dalam sebuah struktur program. Coupling tergantung pada kompleksitas interface antar modul.

Goal: mencoba untuk coupling terendah yang mungkin antar modul.

Coupling yang baik à mengurangi atau mencegah pengaruh perubahan dan efek ripple.
mengurangi biaya pada perubahan program, testing, maintenance

Tipe-tipe coupling:

- data coupling: parameter passing atau interaksi data
- control coupling: share logical kontrol yang berhubungan (untuk sebuah data kontrol)
- common coupling: sharing data umum
- content coupling: modul, penggunaan data atau pengontrolan informasi di-maintain modul lain.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Desain perangkat lunak dan rekayasa perangkat lunak, prinsip desain dan konsep desain

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

12

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Materi ini membahas mengenai pembuatan desain data dan arsitektur sampai dengan pembuatan program

Kompetensi

Mahasiswa dapat membuat desain data, proses data dan pasca pemrosesan data untuk optimasi desain arsitektur dan coding

Desain Perangkat Lunak

1. Desain Data (Data Design)
2. Desain Arsitektur (Architectural Design)
3. Desain Antar Muka (Interface Design)
4. Desain Prosedural (Procedural Design)

1. Desain Data

Desain data adalah aktivitas pertama dan terpenting dari empat aktivitas desain yang dilakukan selama rekayasa perangkat lunak. Proses pemilihan struktur dalam menentukan desain yang paling efisien sesuai kebutuhan. Tujuannya untuk mendapatkan **struktur data** yang baik sehingga diperoleh program yang lebih modular dan mengurangi kompleksitas pengembangan software.

Prinsip Mendesain Data:

- Prinsip analisis sistematis yang diaplikasikan pada fungsi dan perilaku harusnya juga diaplikasikan pada data.
- Semua struktur data dan operasi yang akan dilakukan pada masing-masing struktur data harus diidentifikasi.
- Kamus data harus dibangun dan digunakan untuk menentukan baik data maupun desain program.
- Keputusan desain data tingkat rendah harus ditunda sampai akhir proses desain.
- Representasi struktur data hanya boleh diketahui oleh modul-modul yang menggunakan secara langsung data yang diisikan didalam struktur tersebut.
- Pustaka struktur data dan operasi yang berguna yang dapat diaplikasikan pada struktur data tersebut harus dikembangkan.
- Desain perangkat lunak dan bahasa pemrograman harus mendukung spesifikasi dan realisasi dari tipe-tipe data abstrak.

2. Desain Arsitektur

Desain arsitektur adalah untuk mengembangkan struktur program modular dan merepresentasikan hubungan kontrol antar modul. Metode desain yang disajikan pada bagian ini mendorong prekayasa perangkat lunak untuk berkonsentrasi pada desain arsitektur sebelum mencemaskan masalah perpipaan. Faktor seleksi yang penting untuk

suatu metode desain adalah luasnya aplikasi dimana aplikasi dapat diaplikasikan. Desain berorientasi pada aliran data dapat menyetujui rentang area aplikasi yang luas.

Proses Desain Arsitektur

Desain yang berorientasi pada aliran data merupakan suatu metode desain arsitektur yang memungkinkan transisi yang baik dari model analisis ke deskripsi desain dari struktur program. Transisi dari aliran informasi (yang ditujukan sebagai diagram aliran data) kestruktur dilakukan bagian dari proses 5 langkah:

1. Tipe aliran informasi dibangun.
2. Batas aliran diindikasikan.
3. DFD dipetakan didalam struktur program.
4. Hirarki kontrol ditentukan dengan pemfaktoran.
5. Struktur resultan disaring atau diperhalus dengan menggunakan pengukuran desain dan heuristik.

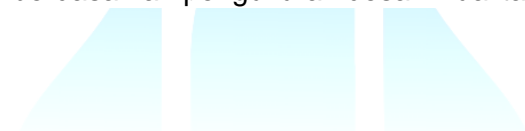
Pasca Pemrosesan Desain

Aplikasi dari pemetaan transaksi dan transformasi yang berhasil kemudian ditambahkan pada dokumentasi tambahan yang dibutuhkan sebagai bagian dari desain arsitektur. Setelah struktur dikembangkan dan disaring, tugas – tugas berikut harus dilakukan:

1. Mengembangkan narasi pemrosesan untuk masing – masing modul.
2. Menyediakan deskripsi interface untuk masing – masing modul.
3. Menentukan struktur data local dan global.
4. Mencatat semua batasan desain.
5. Mengkaji desain.
6. Mempertimbangkan “optimasi” (bila perlu dan dibenarkan).

Optimasi Desain Arsitektur

Desainer perangkat lunak harus memperhatikan perkembangan representasi perangkat lunak yang akan memenuhi semua fungsi dan persyaratan kinerja dan penerimaan jasa berdasarkan pengukuran desain kualitas.



Usul pendekatan berikut ini untuk perangkat lunak kinerja – kritis dalam optimasi desain arsitektur:

1. Kembangkan dan saringlah struktur program tanpa memperhatikan optimasi kinerja – kritis.
2. Gunakan peranti CASE yang mensimulasi kinerja run – time untuk mengisolasi area inefisiensi.
3. selama iterasi desain selanjutnya, pilihlah modul yang dicurigai dan dengan hati – hati kembangkanlah prosedur (algoritma – algoritma) untuk efisiensi waktu.
4. Kodekan sebuah bahasa pemrograman yang sesuai.
5. Instrumentasikan perangkat lunak untuk mengisolasi modul yang menjelaskan utilisasi proses yang berat.
6. Bila perlu, Desain ulang atau kodekan kembali bahasa yang tergantung pada mesin untuk meningkatkan efisiensi.

3. Desain Interface

Memberikan suatu gambaran mengenai struktur program kepada perekraya perangkat lunak. Fokus Desain Interface :

1. Desain interface antar modul
2. Desain interface antara perangkat lunak dan entitas eksternal (produser & konsumen)
3. Desain interface manusia dengan komputer

Desain Interface Manusia-Mesin

Ada empat model yang berbeda pada saat manusia-komputer/ human-komputer interface (HCL) akan didesain. Perekraya perangkat lunak menciptakan sebuah model desain, perekraya perangkat lunak membangun model pemakai, pemakai akhir mengembangkan citra mental yang sering disebut user's model atau perception, dan implementer sistem menciptakan system image.

Model desain dari keseluruhan sistem menggabungkan data, arsitektur, interface, dan representasi prosedural dari perangkat lunak.

Model pemakai menggambarkan profil para pemakai akhir dari sistem. Untuk membangun interface pemakai yang efektif, semua desain harus dimulai dengan suatu pemahaman terhadap pemakai yang dimaksudkan, meliputi profil, usia, jenis kelamin.

Para pemakai juga dapat dikategorikan sebagai:

- Orang baru
- Pemakai intermiten yang banyak pengetahuan
- Pemakai yang banyak pengetahuan dan sering

Persepsi sistem (model pemakai) merupakan citra sistem yang ada dikepala seorang pemakai akhir. Sebagai contoh, bila pemakai pengelola kata tersebut, persepsi sistem akan menuntun respon tersebut.

Citra sistem merangkai manifestasi bagian luar dari sistem berbasis computer (tampilan luar dan “rasa” interface), dengan semua informasi yang mendukung (buku-buku, manual, pita video) yang menggambarkan sintaksis dan semantik sistem.

4. Desain Prosedural

Tujuan: untuk menetapkan detail algoritma yang akan dinyatakan dalam suatu bahasa tertentu. Desain prosedural dilakukan setelah diselesaikannya perancangan desain data, arsitektur, dan antar muka software.

Coding

Program Design Language (PDL)

Bahasa keseluruhan yang sintaksnya dari bahasa tertentu (pemrograman terstruktur).

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman





MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Desain test case, pengujian PL

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

13

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Materi ini membahas mengenai bermacam-macam pengujian perangkat lunak

Kompetensi

Mahasiswa dapat melakukan pengujian Perangkat Lunak dengan mempergunakan berbagai bentuk pengujian.

Pengujian Perangkat Lunak

Pengujian PL adalah elemen kritis dari jaminan kualitas PL dan merepresentasikan spesifikasi, desain dan pengkodean.

Meningkatnya visibilitas PL sebagai suatu elemen sistem dan "biaya" yang muncul akibat kegagalan PL, memotivasi dilakukan perencanaan yang baik melalui pengujian yang teliti.

Dalam melakukan uji coba ada 2 masalah penting yang akan dibahas, yaitu :

- A. Teknik uji coba PL
- B. Strategi uji coba PL

TEKNIK UJI COBA PL

Pada dasarnya, pengujian merupakan suatu proses rekayasa PL yg dapat dianggap (secara psikologis) sebagai hal yg destruktif daripada konstruktif.

SASARAN PENGUJIAN (Glen Myers) :

1. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan.
2. Test case yg baik adalah test case yg memiliki probabilitas tinggi untuk menemukan kesalahan yg belum pernah ditemukan sebelumnya.
3. Pengujian yg sukses adalah pengujian yg mengungkap semua kesalahan yg belum pernah ditemukan sebelumnya.

PRINSIP PENGUJIAN (diusulkan Davis) :

- Semua pengujian harus dapat ditelusuri sampai ke persyaratan pelanggan.
- Pengujian harus direncanakan lama sebelum pengujian itu dimulai.
- Prinsip Pareto berlaku untuk pengujian PL. Prinsip Pareto mengimplikasikan 80% dari semua kesalahan yg ditemukan selama pengujian sepertinya akan dapat ditelusuri sampai 20% dari semua modul program.
- Pengujian harus mulai "dari yg kecil" dan berkembang ke pengujian "yang besar".
- Pengujian yg mendalam tidak mungkin.
- Paling efektif, pengujian dilakukan oleh pihak ketiga yg independen.

TESTABILITAS

Testabilitas PL adalah seberapa mudah sebuah program komputer dapat diuji. Karena pengujian sangat sulit, perlu diketahui apa yg dapat dilakukan untuk membuatnya menjadi mudah.

Karakteristik PL yg diuji :

- OPERABILITAS, semakin baik dia bekerja semakin efisien dia dapat diuji.
- OBSERVABILITAS, apa yg anda lihat adalah apa yg anda uji.

- KONTROLABILITAS, semakin baik kita dapat mengontrol PL semakin banyak pengujian yg dapat diotomatisasi dan dioptimalkan.
- DEKOMPOSABILITAS, dengan mengontrol ruang lingkup pengujian kita dapat lebih cepat mengisolasi masalah dan melakukan pengujian kembali.
- KESEDERHANAAN, semakin sedikit yg diuji semakin cepat pengujian.
- STABILITAS, semakin sedikit perubahan semakin sedikit gangguan pengujian.
- KEMAMPUAN DIPAHAMI, semakin banyak informasi yg dimiliki semakin detail pengujiannya.

ATRIBUT PENGUJIAN YG BAIK :

- Memiliki probabilitas yg tinggi menemukan kesalahan.
- Tidak redundan.
- Harusnya 'jenis terbaik'.
- Tidak boleh terlalu sederhana atau terlalu kompleks.

DESAIN TEST CASE

Terdapat bermacam-macam rancangan metode test case yg dapat digunakan, semua menyediakan pendekatan sistematis untuk uji coba, yg terpenting metode menyediakan kemungkinan yg cukup tinggi menemukan kesalahan.

Terdapat 2 macam test case:

1. Pengetahuan fungsi yg spesifik dari produk yg telah dirancang untuk diperlihatkan, test dapat dilakukan untuk menilai masing-masing fungsi apakah telah berjalan sebagaimana yg diharapkan.
2. Pengetahuan tentang cara kerja dari produk, test dapat dilakukan untuk memperlihatkan cara kerja dari produk secara rinci sesuai dengan spesifikasinya.

Dua macam pendekatan test yaitu :

1. Black Box Testing

Test case ini bertujuan untuk menunjukkan fungsi PL tentang cara beroperasinya, apakah pemasukan data keluaran telah berjalan sebagaimana yang diharapkan dan apakah informasi yang disimpan secara eksternal selalu dijaga kemutakhirannya.

2. White Box Testing

Adalah meramalkan cara kerja perangkat lunak secara rinci, karenanya logikal path (jalur logika) perangkat lunak akan dites dengan menyediakan test case yang akan mengerjakan kumpulan kondisi dan atau pengulangan secara spesifik. Secara sekilas dapat diambil kesimpulan white box testing merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

UJI COBA WHITE BOX

Uji coba white box adalah metode perancangan test case yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan test case. Dengan menggunakan metode white box, analisis sistem akan dapat memperoleh test case yang:

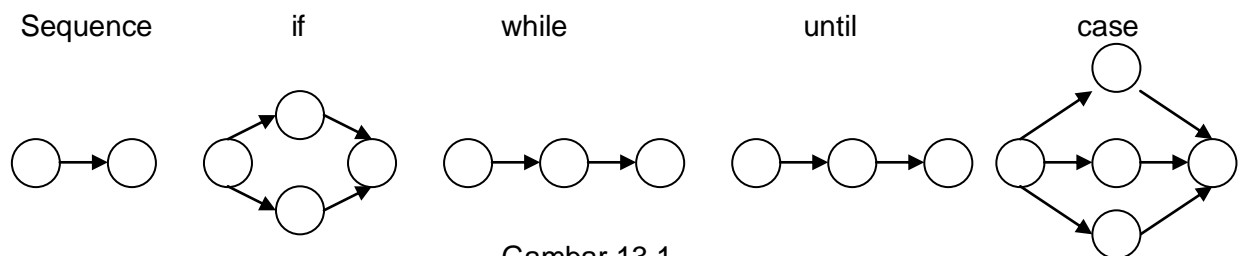
- menjamin seluruh independent path di dalam modul yang dikerjakan sekurang-kurangnya sekali
- mengerjakan seluruh keputusan logikal

- mengerjakan seluruh loop yang sesuai dengan batasannya
- mengerjakan seluruh struktur data internal yang menjamin validitas

1. UJI COBA BASIS PATH

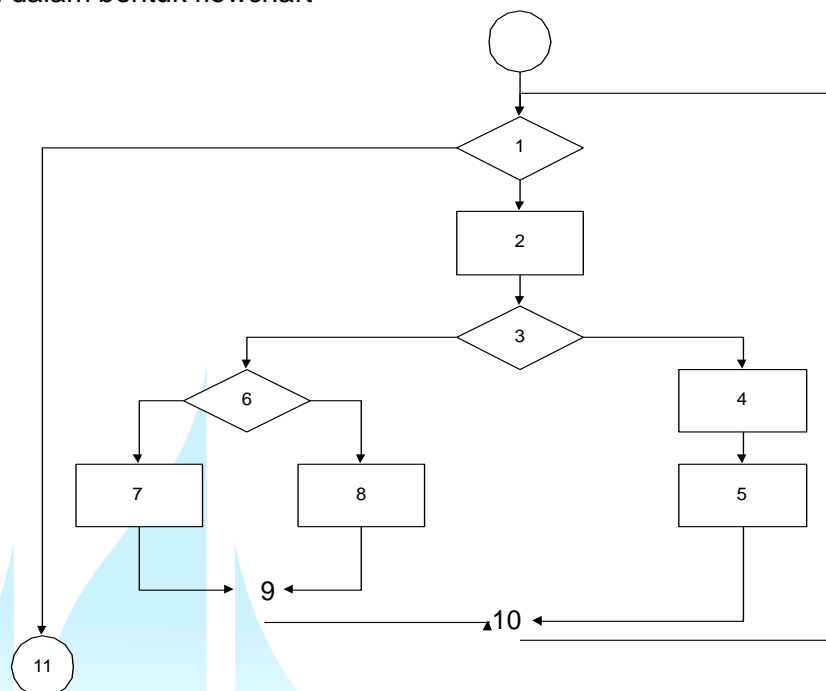
Uji coba basis path adalah teknik uji coba white box yg diusulkan Tom McCabe. Metode ini memungkinkan perancang test case mendapatkan ukuran kekompleksan logical dari perancangan prosedural dan menggunakan ukuran ini sbg petunjuk untuk mendefinisikan basis set dari jalur pengerjaan. Test case yg didapat digunakan untuk mengerjakan basis set yg menjamin pengerjaan setiap perintah minimal satu kali selama uji coba.

1.1. Notasi diagram alir



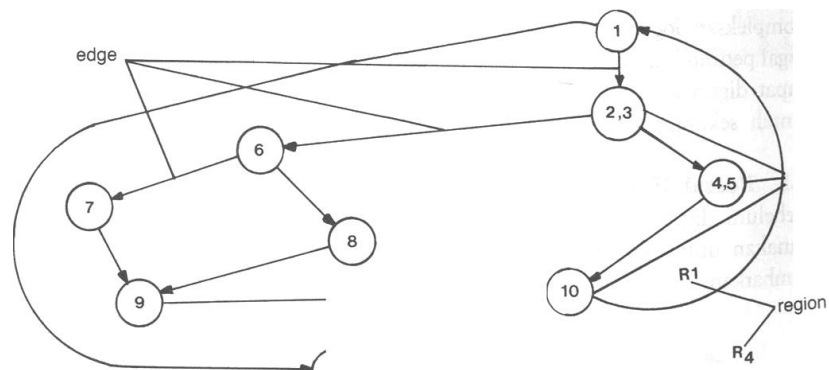
Gambar 13.1

Untuk menggambarkan pemakaian diagram alir diberikan contoh perancangan prosedural dalam bentuk flowchart



Gambar 13.2 Diagram Alir

Selanjutnya diagram alir diatas dipetakan ke grafik alir



Gambar 13.3 Grafik Alir

Lingkaran/node :

menggambarkan satu/lebih perintah prosedural. Urutan proses dan keputusan dapat dipetakan dalam satu node.

Tanda panah/edge :

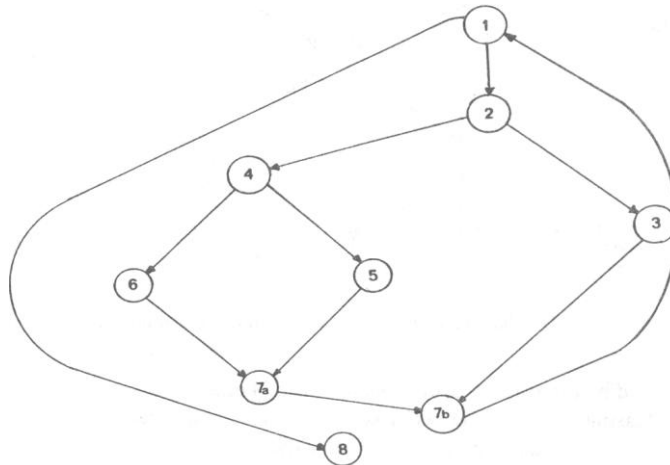
menggambarkan aliran kontrol. Setiap node harus mempunyai tujuan node

Region :

adalah daerah yg dibatasi oleh edge dan node. Termasuk daerah diluar grafik alir.

Contoh menterjemahkan pseudo code ke grafik alir





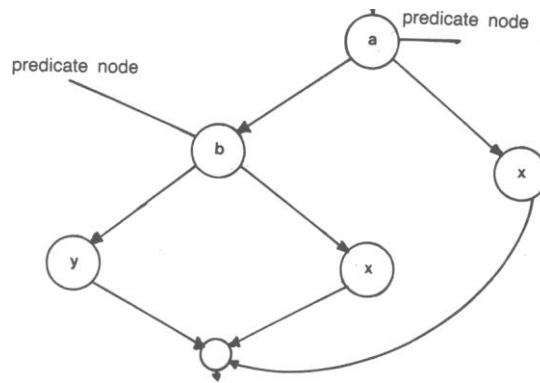
```

1: do while record masih ada
    baca record
2: if record ke 1 = 0
3: then proses record
    simpan di buffer
    naikan kounter
4: else if record ke 2 = 0
5     then reser kounter
6     proses record
        simpan pada file
7a: endif
    endif
7b: enddo
8 : end
  
```

Gambar 13.4 Menerjemahkan PDL ke grafik Alir

Nomor pd pseudo code berhubungan dengan nomor node. Apabila diketemukan kondisi majemuk (compound condition) pada pseudo cade pembuatan grafik alir menjadi rumit. Kondisi majemuk mungkin terjadi pada operator Boolean (AND, OR, NAND, NOR) yg dipakai pada perintah if.

Contoh :



```

if A or B
    then procedure x
    else procedure y
endif
  
```

Gambar 13.5 Logika Gabungan

Node dibuat terpisah untuk masing-masing kondisi A dan B dari pernyataan IF A OR B. Masing-masing node berisi kondisi yg disebut **pridicate node** dan mempunyai karakteristik dua atau lebih edge darinya.

1.2. CYCLOMATIC COMPLEXITY

Cyclomatic complexity adalah metrik PL yang menyediakan ukuran kuantitatif dari kekompleksan logikal program. Apabila digunakan dalam kontek metode uji coba basis path, nilai yang dihitung untuk cyclomatic complexity menentukan jumlah jalur independen dalam basis set suatu program dan memberi batas atas untuk jumlah uji coba yang harus dikerjakan untuk menjamin bahwa seluruh perintah sekurang-kurangnya telah dikerjakan sekali.

Jalur independent adalah jalur yang melintasi atau melalui program dimana sekurang-kurangnya terdapat proses perintah yang baru atau kondisi yang baru.

Dari gambar 9.3 :

Path 1 : 1 - 11

Path 2 : 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11

Path 3 : 1 - 2 - 3 - 6 - 8 - 9 ...: 10 - 1 - 11

Path 4 : 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11

Path 1,2,3,4 yang telah didefinisikan di atas merupakan basis set untuk diagram alir.

Cyclomatic complexity digunakan untuk mencari jumlah path dalam satu flowgraph. Dapat dipergunakan rumusan sbb :

1. Jumlah region grafik alir sesuai dengan cyclomatic complexity.
2. Cyclomatic complexity $V(G)$ untuk grafik alir dihitung dengan rumus:

$$V(G) = E - N + 2$$

dimana:

E = jumlah edge pada grafik alir

N = jumlah node pada grafik alir

3. Cyclomatic complexity $V(G)$ juga dapat dihitung dengan rumus:

$$V(G) = P + 1$$

dimana P = jumlah predicate node pada grafik alir

Pada Gambar 9.3 dapat dihitung cyclomatic complexity:

1. Flowgraph mempunyai 4 region
2. $V(G) = 11 \text{ edge} - 9 \text{ node} + 2 = 4$
3. $V(G) = 3 \text{ predicate node} + 1 = 4$

Jadi cyclomatic complexity untuk flowgraph Gambar 9.3 adalah 4

1.3. MELAKUKAN TEST CASE

Metode uji coba basis path juga dapat diterapkan pada perancangan prosedural rinci atau program sumber. Pada bagian ini akan dijelaskan langkah-langkah uji coba basis path. Prosedur rata-rata pada bagian berikut akan digunakan sebagai contoh dalam pembuatan test case.

PROCEDURE RATA-RATA

INTERFACE RESULT rata, total, input, total.valid

INTERFACE RESULT nilai, minim, max

```

TYPE NILAI (1:100) IS SCALAR ARRAY;
TYPE rata, total. input, total.valid, max.minim, jumlah IS SCALAR;
TYPE I IS INTEGER;

I = 1;

total. input = total. valid = 0;

jumlah = 0;

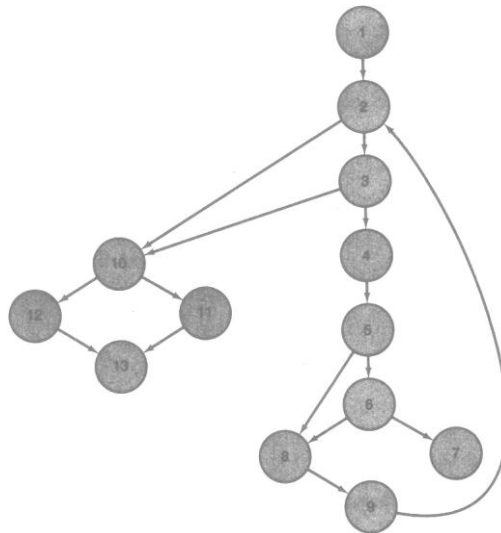
DO WHILE nilai(i) <> -999 .and. total.input < 100
    tambahkan total.input dengan 1;
    IF nilai(i) >= minimum .and. nilai(i) <=max;
    THEN tambahkan total.valid dengan I;
        jumlah=jumlah + nilai(i);
    ELSE skip;
    END IF
    tambahkan i dengan 1;
ENDDO

    IF total. valid> 0
    THEN rata =jumlah/total. valid;
    ELSE rata = -999;
    ENDIF
END

```

Langkah-langkah pembuatan test case:

1. Dengan mempergunakan perancangan prosedural atau program sumber sebagai dasar, digambarkan diagram alirnya.



Gambar 13.6 Diagram Alir prosedur rata

2. Tentukan cyclomatic complexity untuk diagram alir yang telah dibuat:

$$V(G) = 6 \text{ region}$$

$$V(G) = 17 \text{ edge} - 13 \text{ node} + 2 = 6$$

$$V(G) = 5 \text{ predicate node} + 1 = 6$$

3. Tentukan independent path pada flowgraph

Dari hasil perhitungan cyclomatic complexity terdapat 6 independent path yaitu:

path 1 : 1-2-10-11-13

path 2 : 1-2-10-12-13

path 3 : 1-2-3-10-11-13

path 4 : 1-2-3-4-5-8-9-2-..

path 5 : 1-2-3-4-5-6-8-9-2-..

path 6 : 1-2-3-4-5-6-7-8-9-2-...

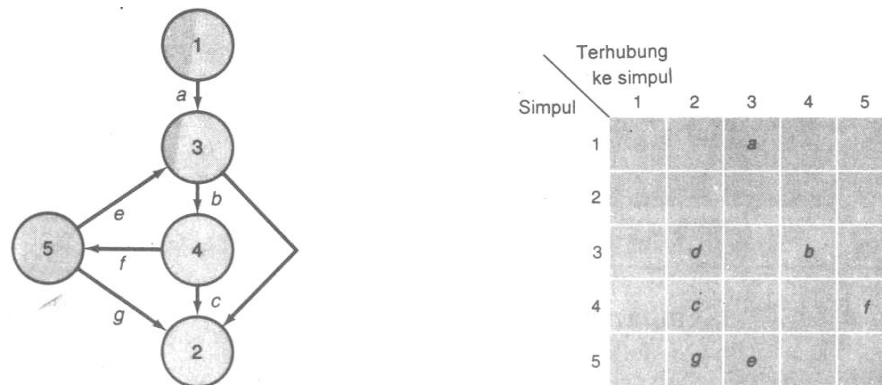
4. Buat test case yang akan mengerjakan masing-masing path pada basis set. Data yang dipilih harus tepat sehingga setiap kondisi dari predicate node dikerjakan semua.

1.4. GRAPH METRIK

Graph metrik merupakan PL yang dikembangkan untuk membantu uji coba basis path atau struktur data. Graph metrik adalah matrik empat persegi yang mempunyai ukuran (sejumlah baris dan kolom) yang sama dengan jumlah node pada flowgraph. Masing-masing baris dan kolom mempunyai hubungan dengan node yang telah ditentukan dan

pemasukan data matrik berhubungan dengan hubungan (edge) antanode.

Contoh sederhana pemakaian graph matrik dapat digambarkan sbb :



Gambar 13.7. Graph matrik

Pada gambar flowgraph masing-masing node ditandai dengan angka dan edge dengan huruf kecil, kemudian diterjemahkan ke graph matrik. Contoh hubungan node 3 dengan node 4 pada graph ditandai dengan huruf b.

Hubungan bobot menyediakan tambahan informasi tentang aliran kontrol. Secara simpel hubungan bobot dapat diberi nilai 1 jika ada hubungan antara node atau nilai 0 jika tidak ada hubungan. Dapat juga hubungan bobot diberi tanda dengan:

- kemungkinan link (edge) dikerjakan
- waktu yang digunakan untuk proses selama traversal dari link
- memori yang diperlukan selama traversal link
- sumber daya yang diperlukan selama traversal link

		Terhubung ke simpul				
		1	2	3	4	5
Simpul	1			1		
	2					
	3		1		1	
	4		1			1
	5		1	1		

Gambar 13.8 Hubungan bobot

Koneksi :

$$1 - 1 = 0$$

$$2 - 1 = 1$$

$$2 - 1 = 1$$

$$2 - 1 = \underline{1}$$

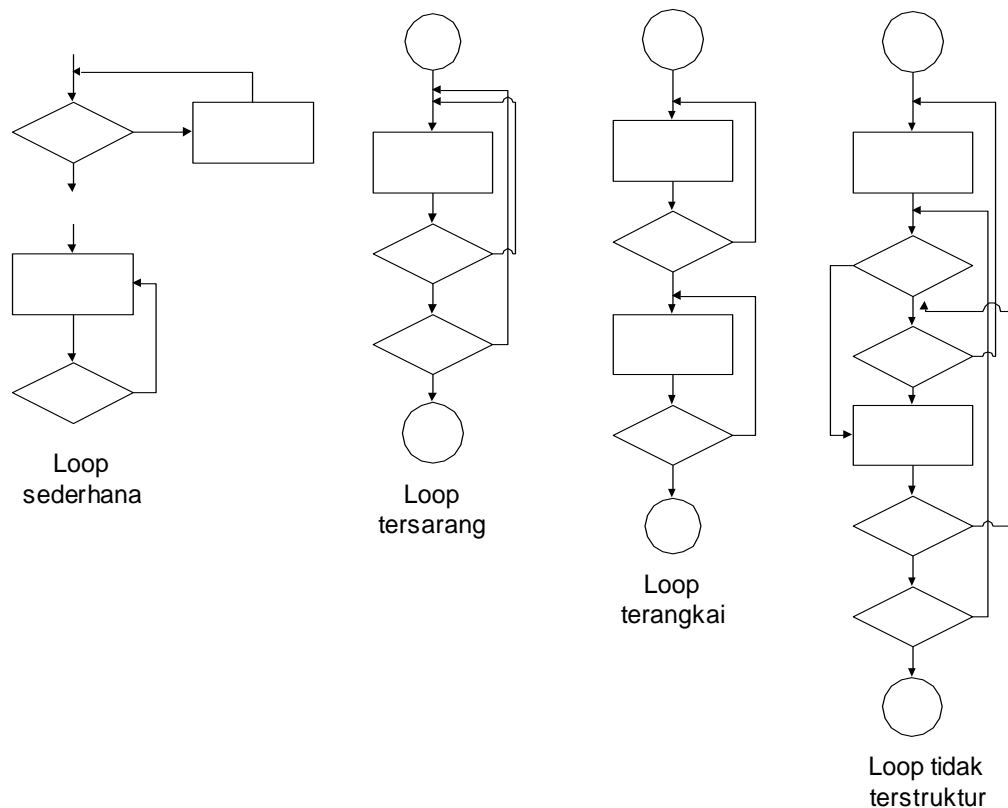
$$3 + 1 = 4 \text{ cyclomatic complexity}$$

2. PENGUJIAN LOOP

Loop merupakan kendala yang sering muncul untuk menerapkan algoritma dengan tepat. Uji coba loop merupakan teknik pengujian white box yg fokusnya pada validitas dari loop.

Kelas loop yaitu :

- a. **Loop Sederhana**, pengujian loop sederhana dilakukan dgn mudah, dimana n jumlah maksimum yg diijinkan melewati loop tsb.
 1. Lewati loop secara keseluruhan
 2. Hanya satu yg dapat melewati loop
 3. m dapat melewati loop dimana $m < n$
- b. **Loop Tersarang**, pengujian loop ini menggunakan pendekatan loop sederhana. Petunjuk pengujian loop tersarang :
 1. Dimulai dari loop paling dalam. Atur semua loop ke nilai minimum.
 2. Kerjakan dgn prinsip loop sederhana untuk loop yg paling dalam sementara tahan loop yg di luar pada parameter terkecil (nilai kounter terkecil)
 3. Kemudian lanjutkan untuk loop yg di atasnya.
 4. Teruskan sampai semua loop selesai di uji.
- c. **Loop Terangkai**, pengujian loop ini menggunakan pendekatan loop sederhana bila masing-masing loop independen, tetapi bila dua loop dirangkai dan pencacah loop 1 digunakan sebagai harga awal loop 2 maka loop tsb jadi tidak independen, maka pendekatan yg diaplikasikan ke loop tersarang direkomendasikan.
- d. **Loop Tidak Terstruktur**, Kapan saja memungkinkan, loop ini didisain kembali agar mencerminkan penggunaan komsepsi pemrograman terstruktur.



Gambar 13.9. Macam-macam loop

PENGUJIAN BLACK-BOX

Pengujian black-box berfokus pada persyaratan fungsional PL. Pengujian ini memungkinkan analisis sistem memperoleh kumpulan kondisi input yang akan mengerjakan seluruh keperluan fungsional program.

Tujuan metode ini mencari kesalahan pada:

- Fungsi yang salah atau hilang
- Kesalahan pada interface
- Kesalahan pada struktur data atau akses database
- Kesalahan performansi
- Kesalahan inisialisasi dan tujuan akhir

Metode ini tidak terfokus pada struktur kontrol seperti pengujian white-box tetapi pada domain informasi.

Pengujian dirancang untuk menjawab pertanyaan sbb:

- Bagaimana validitas fungsional diuji?

- Apa kelas input yg terbaik untuk uji coba yg baik?
- Apakah sistem sangat peka terhadap nilai input tertentu?
- Bagaimana jika kelas data yang terbatas dipisahkan?
- Bagaimana volume data yg dapat ditoleransi oleh sistem?
- Bagaimana pengaruh kombinasi data terhadap pengoperasian system?

1. EQUIVALENCE PARTITIONING

Equivalence partitioning adalah metode pengujian black-box yg memecah atau membagi domain input dari program ke dalam kelas-kelas data sehingga test case dapat diperoleh.

Perancangan test case equivalence partitioning berdasarkan evaluasi kelas equivalence untuk kondisi input yg menggambarkan kumpulan keadaan yg valid atau tidak. Kondisi input dapat berupa nilai numeric, range nilai, kumpulan nilai yg berhubungan atau kondisi Boolean.

Contoh :

Pemeliharaan data untuk aplikasi bank yg sudah diotomatisasikan. Pemakai dapat memutar nomor telepon bank dengan menggunakan mikro komputer yg terhubung dengan password yg telah ditentukan dan diikuti dengan perintah-perintah. Data yg diterima adalah :

Kode area	: kosong atau 3 digit
Prefix	: 3 digit atau tidak diawali 0 atau 1
Suffix	: 4 digit
Password	: 6 digit alfanumerik
Perintah	: check, deposit, dll

Selanjutnya kondisi input digabungkan dengan masing-masing data elemen dapat ditentukan sbb :

Kode area	: kondisi input, Boolean – kode area mungkin ada atau tidak kondisi input, range – nilai ditentukan antara 200 dan 999
Prefix	: kondisi input range > 200 atau tidak diawali 0 atau 1

Suffix : kondisi input nilai 4 digit

Password : kondisi input boolean – pw mungkin diperlukan atau tidak
kondisi input nilai dengan 6 karakter string

Perintah : kondisi input set berisi perintah-perintah yang telah didefinisikan

2. BOUNDARY VALUE ANALYSIS

Untuk permasalahan yg tidak diketahui dg jelas cenderung menimbulkan kesalahan pada domain outputnya. BVA merupakan pilihan test case yg mengerjakan nilai yg telah ditentukan, dgn teknik perancangan test case melengkapi test case equivalence partitioning yg fokusnya pada domain input. BVA fokusnya pada domain output.

Petunjuk pengujian BVA :

1. Jika kondisi input berupa range yg dibatasi nilai a dan b, test case harus dirancang dgn nilai a dan b.
2. Jika kondisi input ditentukan dgn sejumlah nilai, test case harus dikembangkan dgn mengerjakan sampai batas maksimal nilai tsb.
3. Sesuai petunjuk 1 dan 2 untuk kondisi output dirancang test case sampai jumlah maksimal.
4. Untuk struktur data pada program harus dirancang sampai batas kemampuan.

STRATEGI PENGUJIAN PL

Strategi uji coba PL memudahkan para perancang untuk menentukan keberhasilan system yg telah dikerjakan. Hal yg harus diperhatikan adalah langkah-langkah perencanaan dan pelaksanaan harus direncanakan dengan baik dan berapa lama waktu, upaya dan sumber daya yg diperlukan.

Strategi uji coba mempunyai karakteristik sbb :

- Pengujian mulai pada tingkat modul yg paling bawah, dilanjutkan dgn modul di atasnya kemudian hasilnya dipadukan.
- Teknik pengujian yang berbeda mungkin menghasilkan sedikit perbedaan (dalam hal waktu)
- Pengujian dilakukan oleh pengembang perangkat lunak dan (untuk proyek yang besar) suatu kelompok pengujian yang independen.
- Pengujian dan debugging merupakan aktivitas yang berbeda, tetapi debugging termasuk dalam strategi pengujian.

Pengujian PL adalah satu elemen dari topik yang lebih luas yang sering diacu sebagai *verifikasi dan validasi (V&V)*.

Verifikasi : Kumpulan aktifitas yg menjamin penerapan PL benar-benar sesuai dgn

fungsinya.

Validasi : Kumpulan aktivitas yang berbeda yang memastikan bahwa PL yang dibangun dapat memenuhi keperluan pelanggan.

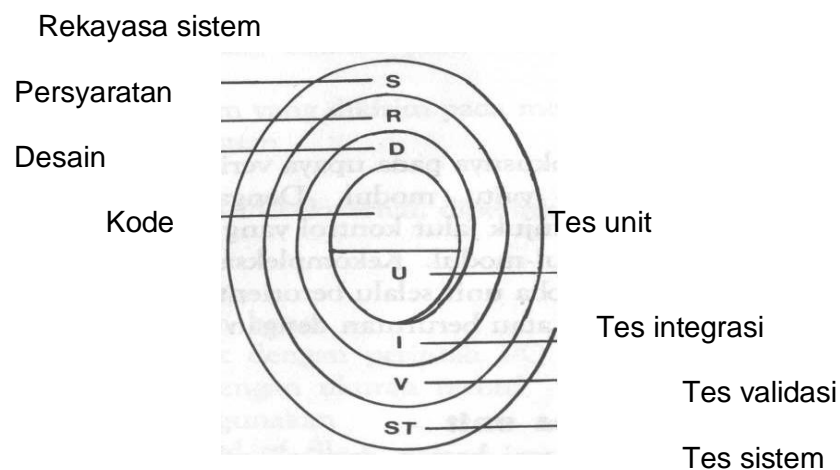
Dgn kata lain :

Verifikasi : “ Apakah kita membuat produk dgn benar?”

Validasi : “ Apakah kita membuat benar-benar suatu produk?”

Definisi dari V&V meliputi berbagai aktivitas yang kita rujuk sebagai jaminan kualitas PL (SQA).

Pengujian merupakan salah satu tugas yg ada dlm arus siklus pengembangan system yg dapat digambarkan dalam bentuk spiral :



Gambar 9.10. Strategi Uji Coba

1. PENGUJIAN UNIT

Unit testing (uji coba unit) fokusnya pada usaha verifikasi pada unit terkecil dari desain PL, yakni modul. Uji coba unit selalu berorientasi pada white box testing dan dapat dikerjakan paralel ayau beruntun dengan modul lainnya.

1.1 Pertimbangan Pengujian Unit

Interface diuji cobakan untuk menjamin informasi yg masuk atau yg ke luar dari unit program telah tepat atau sesuai dgn yg diharapkan. Yg pertama diuji coba adalah interface karena diperlukan untuk jalannya informasi atau data antar modul.

Myers mengusulkan checklist untuk pengujian interface:

- Apakah jumlah parameter input sama dengan jumlah argumen?
- Apakah antara atribut dan parameter argumen sudah cocok?
- Apakah antara sistem satuan parameter dan argumen sudah cocok?
- Apakah jumlah argumen yang ditransmisikan ke modul yang dipanggil sama dengan jumlah parameter?
- Apakah atribut dari argumen yang ditransmisikan ke modul yang dipanggil sama dengan atribut parameter?
- Apakah sistem unit dari argumen yang ditransmisikan ke modul yang dipanggil sama dengan sistem satuan parameter?
- Apakah jumlah atribut dari urutan argumen ke fungsi-fungsi built-in sudah benar?
- Adakah referensi ke parameter yang tidak sesuai dengan pain entri yang ada?
- Apakah argumen input-only diubah?
- Apakah definisi variabel global konsisten dengan modul?
- Apakah batasan yang dilalui merupakan argumen?

Bila sebuah modul melakukan I/O eksternal, maka pengujian interface tambahan harus dilakukan.

- Atribut file sudah benar?
- Pernyataan OPEN/CLOSE sudah benar?
- Spesifikasi format sudah cocok dengan pernyataan I/O?
- Ukuran buffer sudah cocok dengan ukuran rekaman?
- File dibuka sebelum penggunaan?
- Apakah kondisi End-of-File ditangani?
- Kesalahan I/O ditangani?
- Adakah kesalahan tekstual di dalam informasi output?

Kesalahan yang umum di dalam komputasi adalah:

- kesalahan-pahaman atau prosedur aritmatik yang tidak benar
- operasi mode yang tercampur
- inisialisasi yang tidak benar

- inakurasi ketelitian
- representasi simbolis yang tidak benar dari sebuah persamaan.

Test case harus mengungkap kesalahan seperti

- perbandingan tipe data yang berbeda
- preseden atau operator logika yang tidak benar
- pengharapan akan persamaan bila precision error membuat persamaan yang tidak mungkin
- perbandingan atau variabel yang tidak benar
- penghentian loop yang tidak ada atau tidak teratur
- kegagalan untuk keluar pada saat terjadi iterasi divergen
- variabel loop yang dimodifikasi secara tidak teratur.

1.2. Prosedur Pengujian Unit

Program sumber telah dikembangkan, ditunjang kembali dan diverifikasi untuk sintaksnya, maka perancangan test case dimulai. Peninjauan kembali perancangan informasi akan menyediakan petunjuk untuk menentukan test case. Karena modul bukan program yg berdiri sendiri maka driver (pengendali) dan atau stub PL harus dikembangkan untuk pengujian unit.

Driver adl program yg menerima data untuk test case dan menyalurkan ke modul yg diuji dan mencetak hasilnya.

Stub melayani pemindahan modul yg akan dipanggil untuk diuji.

2. PENGUJIAN INTEGRASI

Pengujian terintegrasi adl teknik yg sistematis untuk penyusunan struktur program, pada saat bersamaan dikerjakan uji coba untuk memeriksa kesalahan yg nantinya digabungkan dengan interface.

Metode pengujian

- top down integration
- bottom up integration

2.1. TOP DOWN INTEGRATION

Merupakan pendekatan inkrmmental untuk penyusunan struktur program. Modul dipadukan dgn bergerak ke bawah melalui kontrol hirarki dimulai dari modul utama.

Modul subordinat ke modul kontrol utama digabungkan ke dalam struktur baik menurut depth first atau breadth first.

Proses integrasi:

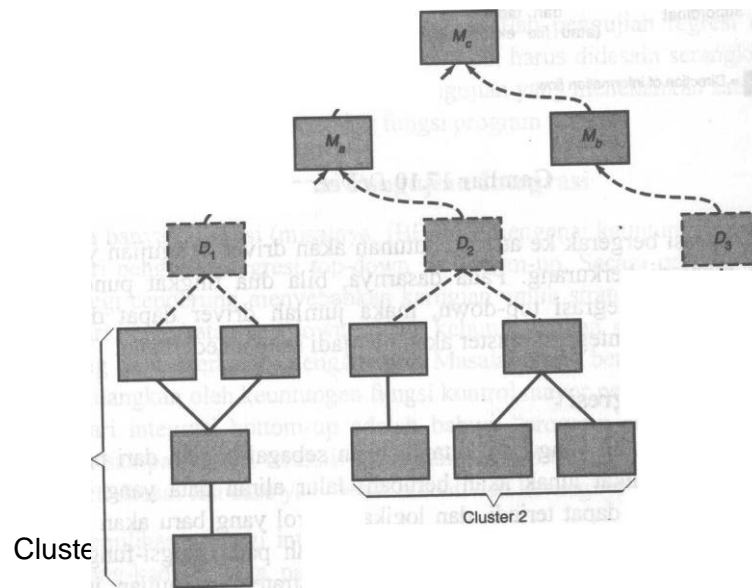
- modul utama digunakan sebagai test driver dan stub yg menggantikan seluruh modul yg secara langsung berada di bawah modul kontrol utama.
- Tergantung pada pendekatan perpaduan yg dipilih (depth / breadth)
- Uji coba dilakukan selama masing-masing modul dipadukan
- Pada penyelesaian masing-masing uji coba stub yg lain dipindahkan dgn modul sebenarnya.
- Uji coba regression yaitu pengulangan pengujian untuk mencari kesalahan lain yg mungkin muncul.

2.2. BOTTOM UP INTEGRATION

Pengujian bottom up dinyatakan dgn penyusunan yg dimulai dan diujicobakan dgn atomic modul (yi modul tingkat paling bawah pd struktur program). Karena modul dipadukan dari bawah ke atas, proses yg diperlukan untuk modul subordinat yg selalu diberikan harus ada dan diperlukan untuk stub yg akan dihilangkan.

Strategi pengujian :

- Modul tingkat bawah digabungkan ke dalam cluster yg memperlihatkan subfungsi PL
- Driver (program kontrol pengujian) ditulis untuk mengatur input test case dan output
- Cluster diuji
- Driver diganti dan cluster yg dikombinasikan dipindahkan ke atas pada struktur program



Gambar 13.11. Bottom Up Integration

3. UJI COBA VALIDASI

Setelah semua kesalahan diperbaiki maka langkah selanjutnya adalah validasi terting. Pengujian validasi dikatakan berhasil bila fungsi yg ada pada PL sesuai dgn yg diharapkan pemakai.

Validasi PL merupakan kumpulan seri uji coba black box yg menunjukkan sesuai dgn yg diperlukan.

Kemungkinan kondisi setelah pengujian:

1. Karakteristik performansi fungsi sesuai dgn spesifikasi dan dapat diterima.
2. Penyimpangan dari spesifikasi ditemukan dan dibuatkan daftar penyimpangan.

Pengujian BETA dan ALPHA

Apabila PL dibuat untuk pelanggan maka dapat dilakukan acceptance test sehingga memungkinkan pelanggan untuk memvalidasi seluruh keperluan. Test ini dilakukan karena memungkinkan pelanggan menemukan kesalahan yg lebih rinci dan membiasakan pelanggan memahami PL yg telah dibuat.

Pengujian Alpha

Dilakukan pada sisi pengembang oleh seorang pelanggan. PL digunakan pada setting yg natural dgn pengembang “yg memandang” melalui bahu pemakai dan merekam semua kesalahan dan masalah pemakaian.

Pengujian Beta

Dilakukan pada satu atau lebih pelanggan oleh pemakai akhir PL dalam lingkungan yg sebenarnya, pengembang biasanya tidak ada pada pengujian ini. Pelanggan merekan semua masalah (real atau imajiner) yg ditemui selama pengujian dan melaporkan pada pengembang pada interval waktu tertentu.

4. UJI COBA SISTEM

Pada akhirnya PL digabungkan dgn elemen system lainnya dan rentetan perpaduan system dan validasi tes dilakukan. Jika uji coba gagal atau di luar skope dari proses daur siklus pengembangan system, langkah yg diambil selama perancangan dan pengujian dapat diperbaiki. Keberhasilan perpaduan PL dan system yg besar merupakan kuncinya.

Sistem testing merupakan rentetan pengujian yg berbeda-beda dgn tujuan utama mengerjakan keseluruhan elemen system yg dikembangkan.

4.1. Recovery Testing

Adalah system testing yg memaksa PL mengalami kegagalan dalam bermacam-macam cara dan memeriksa apakah perbaikan dilakukan dgn tepat.

4.2. Security Testing

Adalah pengujian yg akan melakukan verifikasi dari mekanisme perlindungan yg akan dibuat oleh system, melindungi dari hal-hal yg mungkin terjadi.

4.3. Strees Testing

Dirancang untuk menghadapi situasi yg tidak normal pada saat program diuji. Testing ini dilakukan oleh system untuk kondisi seperti volume data yg tidak normal (melebihi atau kurang dari batasan) atau frkkuensi.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Konsep pemeliharaan Perangkat Lunak

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

14

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Materi ini membahas mengenai Konsep pemeliharaan perangkat lunak dengan teknik pemeliharaan perangkat lunak.

Kompetensi

Mahasiswa dapat memahami konsep pemeliharaan perangkat lunak dalam melakukan rekayasa dengan mempergunakan teknik pemeliharaan yang dibutuhkan.

Konsep pemeliharaan PL

Istilah pemeliharaan perangkat lunak digunakan untuk menjabarkan aktivitas dari analisis sistem (software engineering) yang terjadi pada saat hasil produk perangkat lunak sudah dipergunakan oleh pemakai (user).

Biasanya pengembangan produk perangkat lunak memerlukan waktu antara 1 sampai dengan 2 tahun, tetapi pada fase pemeliharaan perangkat lunak menghabiskan 5 sampai dengan 10 tahun. Aktivitas yang terjadi pada fase pemeliharaan antara lain:

- Penambahan atau peningkatan atau juga perbaikan untuk produk perangkat lunak
- Adaptasi produk dengan lingkungan mesin yang baru
- Pembetulan permasalahan yang timbul

“ Pemeliharaan sistem berawal begitu sistem baru menjadi operasional dan berakhir masa hidupnya ”

Jenis Pemeliharaan :

- Pemeliharaan Korektif : Pemeliharaan perangkat lunak dengan melakukan perbaikan kesalahan yang terjadi pada perangkat lunak
- Pemeliharaan Adaptif : Pemeliharaan perangkat lunak dengan melakukan penyesuaian fungsi-fungsi yang ada pada perangkat lunak sehingga lebih memudahkan user.
- Pemeliharaan Penyempurnaan : Pemeliharaan perangkat lunak dengan melakukan pengembangan / peningkatan terhadap perangkat lunak yang telah ada.
- Pemeliharaan Preventif : Pemeliharaan perangkat lunak dengan perombakan secara total atau melakukan perekayasa kembali pada perangkat lunak yang ada.

Siklus Hidup Pemeliharaan Sistem (SMLC)

Tahapan SMLC :

- Memahami Permintaan Pemeliharaan
- Mentransformasi permintaan pemeliharaan menjadi perubahan
- Menspesifikasi perubahan
- Mengembangkan perubahan

- Menguji perubahan
- Melatih pengguna dan melakukan test penerimaan
- Pengkonversian dan meluncurkan operasi
- Mengupdate Dokumen
- Melakukan pemeriksaan Pasca implementasi

Maintainability (Kemampuan pemeliharaan sistem)

Prosedur untuk peningkatan maintainability :

- Menerapkan SDLC dan SWDLC
- Menspesifikasi definisi data standar
- Menggunakan bahasa pemrograman standart
- Merancang modul-modul yang terstruktur dengan baik
- Mempekerjakan modul yang dapat digunakan kembali
- Mempersiapkan dokumentasi yang jelas, terbaru dan komprehensif
- Menginstall perangkat lunak, dokumentasi dan soal-soal test di dalam sentral repositor sistem CASE atau CMS (change management system)

Tiga pendekatan untuk menyusun Pemeliharaan sistem :

- Pendekatan Pemisahan -> Pemeliharaan dan Pemeliharaan
- Pendekatan Gabungan -> Menggabungkan personalia penyusun dan pemelihara menjadi sebuah kelompok utama sistem informasi
- Pendekatan Fungsional -> Variasi dari pendekatan gabungan dengan memindahkan tenaga profesional sistem dari sistem informasi dan menugasi mereka pada fungsi bisnis untuk penyusunan maupun pemeliharaan.

Ada 5 CASE Tools yang membantu pemeliharaan sistem dari sistem lama dan membantu memecahkan kemacetan timbunan sistem baru yang belum dikerjakan :

- Rekayasa Maju (Forward engineering)
- Rekayasa Mundur (Reverse engineering)
- Rekayasa Ulang (Reengineering)
- Restrukturisasi (restructuring)
- Sistem Pakar Pemeliharaan (Maintenance expert system)

Mengelola Pemeliharaan Sistem

- Menetapkan Kegiatan Pemeliharaan Sistem

- Mengawali dan merekam kegiatan pemeliharaan sistem tidak terjadwal (Form Maintenance Work Order : Pekerjaan yang diperlukan/dilakukan, waktu yang diperkirakan dibandingkan dengan waktu yang sebenarnya, kode pemeliharaan, biaya pemeliharaan)
- Menggunakan sistem perangkat lunak helpdesk
- Mengevaluasi aktivitas pemeliharaan sistem
- Mengoptimalkan program pemeliharaan sistem

Martin & McClure (1983)

- ...perubahan yang harus dilakukan terhadap program komputer setelah diserahkan kepada pelanggan / pengguna

Von Mayrhauser (1990)

- Pemeliharaan mencakup hidup sistem perangkat lunak sejak diinstalasi sampai tidak digunakan lagi

Martin & McClure (1993)

Pemeliharaan P/L dilakukan dengan maksud tertentu

- Memperbaiki kesalahan
- Memperbaiki kekurangan pada rancangan
- Interaksi dengan sistem lain
- Perluasan sistem
- Melakukan perubahan yang perlu
- Melakukan perubahan file atau basisdata
- Meningkatkan rancangan
- Mengubah program untuk dapat memanfaatkan P/K, P/L, fitur sistem, fasilitas komunikasi.

Semua sistem informasi sewaktu-waktu berubah. Pemeliharaan sistem adalah kegiatan yang membuat perubahan ini.

KEPERLUAN PEMELIHARAAN SISTEM

Sistem perlu dipelihara karena beberapa hal, yaitu :

1. Sistem memiliki kesalahan yang dulunya belum terdeteksi, sehingga kesalahan-kesalahan sistem perlu diperbaiki.

2. Sistem mengalami perubahan-perubahan karena permintaan baru dari pemakai sistem.
3. Sistem mengalami perubahan karena perubahan lingkungan luar (perubahan bisnis).
4. Sistem perlu ditingkatkan.

Biaya pemeliharaan sistem sering diabaikan. Kenyataannya biaya pemeliharaan sistem merupakan biaya yang cukup besar.

Biaya pemeliharaan perangkat lunak telah terus menerus naik selama 25 tahun terakhir.

Beberapa perusahaan membelanjakan 80% atau lebih dari anggaran sistem mereka pada pemeliharaan perangkat lunak.

JENIS PEMELIHARAAN SISTEM

Pemeliharaan sistem dapat digolongkan menjadi empat jenis :

- Pemeliharaan Korektif
- Pemeliharaan Adaptif
- Pemeliharaan Perfektif (Penyempurnaan)
- Pemeliharaan Preventif

Pemeliharaan Korektif

Pemeliharaan korektif adalah bagian pemeliharaan sistem yang tidak begitu tinggi nilainya dan lebih membebani, karena pemeliharaan ini mengoreksi kesalahan-kesalahan yang ditemukan pada saat sistem berjalan.

Umumnya pemeliharaan korektif ini mencakup kondisi penting atau bahaya yang memerlukan tindakan segera. Kemampuan untuk mendiagnosa atau memperbaiki kesalahan atau malfungsi dengan cepat sangatlah berharga bagi perusahaan.

Pemeliharaan Adaptif

Pemeliharaan adaptif dilakukan untuk menyesuaikan perubahan dalam lingkungan data atau pemrosesan dan memenuhi persyaratan pemakai baru.

Lingkungan tempat sistem beroperasi adalah dinamik, dengan demikian, sistem harus terus merespon perubahan persyaratan pemakai. Misalnya, Undang-Undang Perpajakan yang baru mungkin memerlukan suatu perubahan dalam kalkulasi pembayaran bersih.

Umumnya pemeliharaan adatif ini baik dan tidak dapat dihindari.

Pemeliharaan Penyempurnaan

Pemeliharaan penyempurnaan mempertinggi cara kerja atau maintainabilitas (kemampuan untuk dipelihara). Tindakan ini juga memungkinkan sistem untuk memenuhi persyaratan pemakai yang sebelumnya tidak dikenal.

Ketika membuat perubahan substansial modul apapun, petugas pemeliharaan juga menggunakan kesempatan untuk mengupgrade kode, mengganti cabang-cabang yang kadaluwarsa, memperbaiki kecerobohan, dan mengembangkan dokumentasi.

Sebagai contoh, kegiatan pemeliharaan ini dapat berbentuk perekayasaan ulang atau restrukturisasi perangkat lunak, penulisan ulang dokumentasi, pengubahan format dan isi laporan, penentuan logika pemrosesan yang lebih efisien, dan pengembangan efisiensi pengoperasian perangkat.

Pemeliharaan Preventif

Pemeliharaan Preventif terdiri atas inspeksi periodik dan pemeriksaan sistem untuk mengungkap dan mengantisipasi permasalahan.

Karena personil pemeliharaan sistem bekerja dalam sistem ini, mereka seringkali menemukan cacat-cacat (bukan kesalahan yang sebenarnya) yang menandakan permasalahan potensial. Sementara tidak memerlukan tindakan segera, cacat ini bila tidak dikoreksi di tingkat awal, jelas sekali akan mempengaruhi baik fungsi sistem maupun kemampuan untuk memeliharanya dalam waktu dekat.

Daftar Pustaka

- Software Engineering Ian Sommerville
- Software Engineering Roger S.Pressman



MODUL PERKULIAHAN

Rekayasa Perangkat Lunak

Sistematika Dokumentasi

Fakultas
Ilmu Komputer

Program Studi
Sistem Informasi

Tatap Muka

15

Kode MK
87011

Disusun Oleh
Tim Dosen

Abstract

Materi ini membahas mengenai sistematika dokumentasi kegiatan rekayasa lunak

Kompetensi

Mahasiswa dapat memahami tata cara dokumentasi dari tahap perancangan sampai dengan implementasi

Sistematika dokumentasi

Menurut Roger S. Pressman [1], ada tiga hal yang dapat mendefinisikan suatu perangkat lunak yaitu:

(1) program komputer yang bila dieksekusi akan memberikan fungsi dan kerja seperti yang diinginkan.

(2) struktur data yang memungkinkan program memanipulasi informasi secara proposional, dan

(3) dokumen yang menggambarkan operasi dan kegunaan program. Sehingga dapat dikatakan sebuah program komputer belum dapat disebut perangkat lunak tanpa disertai dengan dokumentasinya. Hal ini menunjukkan betapa pentingnya dokumentasi pada pembuatan sebuah perangkat lunak, tetapi banyak pengembang perangkat lunak yang kurang memperhatikan masalah dokumentasi.

Dokumentasi merupakan sebuah artefak yang tujuannya untuk menyampaikan informasi tentang sistem perangkat lunak yang menyertainya. Selain itu dokumentasi mempunyai fungsi sebagai berikut :

1. Bertindak sebagai media komunikasi antar anggota pengembang tim,
2. Penyimpanan sistem informasi untuk digunakan oleh maintenance engineers,
3. Membantu manajer proyek dalam merencanakan, mengatur anggaran, dan penjadwalan dalam proses pembangunan perangkat lunak,
4. Memberi penjelasan kepada pengguna bagaimana cara menggunakan dan mengelola sistem yang dibangun.

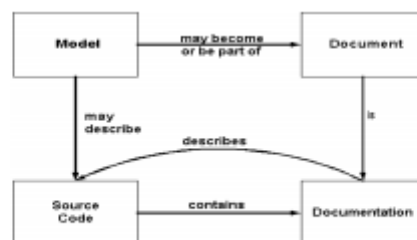
Sebagai tempat penyimpanan informasi, dokumen semestinya harus berisi informasi yang lengkap, valid, mudah dimengerti, dan up-to-date. Tapi sayangnya banyak pengembang yang membiarkan dokumen yang dibuat tidak memberikan informasi yang lengkap atau informasi yang tidak diperbaharui (out-of-date).

Beberapa software engineers berpendapat bahwa “my code is self-documenting”.

Mereka beranggapan cukup dengan source code sudah merupakan dokumentasinya, sehingga tidak diperlukan dokumen tambahan. Hal ini mungkin dapat berlaku jika program yang dibuat untuk dirinya sendiri. Tetapi bagaimana jika program tersebut digunakan oleh orang lain

atau program tersebut sebagai bagian dari sebuah sistem perangkat lunak yang dikerjakan oleh banyak orang ? Software engineers yang lain mungkin dapat mengerti jalannya program dengan membaca kode tersebut, tetapi tetap akan membutuhkan waktu yang lebih lama dibandingkan dengan membaca sebuah dokumen yang menjelaskan secara rinci tentang program tersebut. Scott Ambler dalam thesis Andrew Forward menjelaskan hubungan antara source code, model, dokumen, dan dokumentasi . Ambler menjelaskan bahwa sebuah dokumentasi merupakan penjelasan dari kode yang dibuat. Sebuah model juga mungkin menjelaskan kode, dan model ini dapat menjadi dokumen atau bagian dari dokumen.

Hubungan tersebut dapat dilihat pada gambar berikut



Gambar 1: Hubungan antara *source code*, *model*, *document*, dan *documentation*

Ian Sommerville mengklasifikasi dokumentasi ke dalam dua kelas, yaitu dokumentasi proses dan dokumentasi produk .

Dokumentasi proses merupakan dokumen yang menyimpan semua proses dari pembangunan dan pemeliharaan perangkat lunak, termasuk perencanaan, penjadwalan, lembar kerja, serta memo maupun email.

Sedangkan dokumen produk yaitu dokumen yang merupakan penjelasan dari perangkat lunak yang dibangun. Dokumentasi pengguna dan dokumentasi sistem termasuk dalam dokumen produk. Dokumentasi pengguna yaitu dokumen yang menjelaskan tentang bagaimana penggunaan dari produk perangkat lunak tersebut, sedangkan dokumen sistem yaitu semua dokumen yang menjelaskan tentang sistem yang dibangun, mulai dari spesifikasi kebutuhan sampai dengan pengujian perangkat lunak.

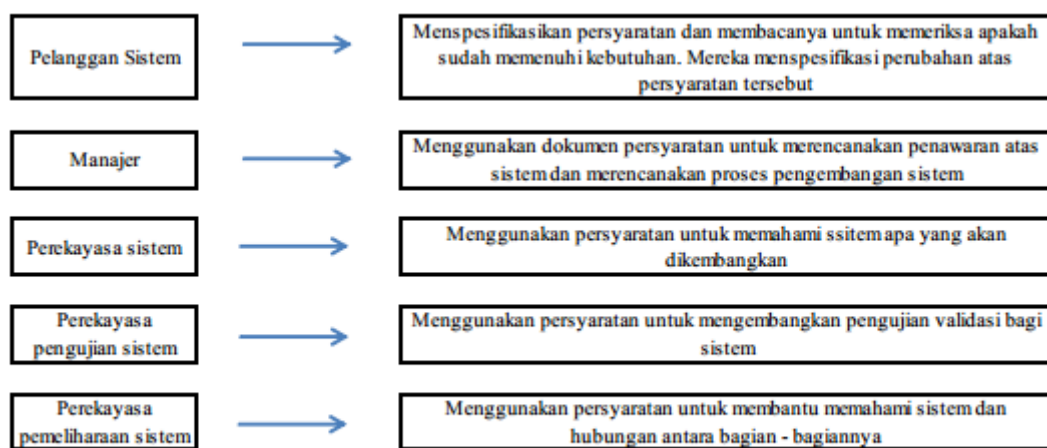
Pada sumber lain ada yang mengklasifikasikan dokumentasi ke dalam empat bagian yaitu dokumen kebutuhan, arsitektur dan desain, dokumen teknis, dokumen end user, dan dokumen pemasaran . Dokumen kebutuhan merupakan dokumen yang menjelaskan tentang atribut, kemampuan, karakteristik, atau kualitas dari suatu sistem yang merupakan dasar dari

pembuatan suatu perangkat lunak. Dokumen arsitektur dan disain yaitu dokumen yang menjelaskan tentang arsitektur sistem dan prinsip – prinsip konstruksi yang akan digunakan dalam desain komponen perangkat lunak. Dokumen teknis merupakan dokumentasi dari kode, algoritma dan interface. Dokumen end user merupakan dokumen manual tentang bagaimana perangkat lunak tersebut digunakan. Dokumen pemasaran berisi bagaimana cara pemasaran dari produk dan analisis permintaan pasar.

2.1 Dokumen Persyaratan Perangkat Lunak

Dokumen persyaratan perangkat lunak (SRS/Software Requirements Specification) merupakan persyaratan resmi mengenai apa yang dituntut dari pengembang sistem [7].

Dokumen berisi persyaratan user untuk sistem dan spesifikasi secara rinci dari persyaratan sistem. Berikut ilustrasi contoh dokumen persyaratan perangkat lunak dan bagaimana pemanfaatannya [7].



Gambar 2: Ilustrasi pemanfaatan dokumen persyaratan perangkat lunak

Heninger dalam buku Ian Sommerville [7] mengusulkan bahwa ada enam persyaratan yang harus dipenuhi oleh dokumen persyaratan perangkat lunak yaitu:

- Menspesifikasi perilaku sistem eksternal.
- Menspesifikasi batasan – batasan implementasi.
- Mudah diubah
- Berfungsi sebagai alat bantu referensi bagi pemelihara sistem.

Lembaga IEEE telah membuat standar untuk dokumen persyaratan perangkat lunak (IEEE/ANSI 830-1993). Berikut outline yang disarankan oleh IEEE untuk dokumen persyaratan perangkat lunak:

1. **Pendahuluan**
 - 1.1 Tujuan dokumen persyaratan
 - 1.2 Cakupan produk
 - 1.3 Definisi, akronim, dan singkatan
 - 1.4 Referensi
 - 1.5 Tinjauan bagian dokumen berikutnya
2. **Deskripsi umum**
 - 2.1 Perspektif produk
 - 2.2 Fungsi produk
 - 2.3 Karakteristik user
 - 2.4 Batasan-batasan umum
 - 2.5 Asumsi dan ketergantungan
3. **Persyaratan khusus**
4. **Lampiran**
5. **Indeks**

Gambar 3: Outline dokumen persyaratan perangkat lunak [7]

Persyaratan khusus mencakup persyaratan fungsional, non-fungsional dan interface yang merupakan bagian penting dari dokumen persyaratan perangkat lunak. Standar dari IEEE memberikan saran apa saja yang perlu ditulis di dokumen persyaratan perangkat lunak, tetapi pemanfaatannya tergantung dari kebutuhan pengembang dan pengguna perangkat lunak tersebut.

2.2 Dokumentasi Desain

Dokumentasi desain berisi penjelasan rinci tentang inti teknis dari rekayasa perangkat lunak yang meliputi struktur data, arsitektur program, interface dan detail prosedural [1].

Gambar 3 menunjukkan contoh outline dari dokumen desain yang diambil dari buku Pressman [1]. Berikut penjelasan perbagian dari Pressman mengenai outline tersebut:

- ☐ Bagian I berisi ruang lingkup dari kerja desain.
- ☐ Bagian II berisi desain data, struktur file eksternal dan referensi silang yang menghubungkan objek data dengan file tertentu.
- ☐ Bagian III berisi desain arsitektur.
- ☐ Bagian IV dan V, pada bagian ini berkembang pada saat desain interface dan procedural dimulai.
- ☐ Bagian VI berisi referensi silang yang bertujuan untuk menetapkan bahwa semua persyaratan dipenuhi oleh desain perangkat lunak dan menunjukkan modul mana yang kritis terhadap implementasi persyaratan spesifik.
- ☐ Bagian VII berisi tahap pertama dari pembuatan dokumentasi pengujian.
- ☐ Bagian VIII dan IX berisi data tambahan meliputi deskripsi algoritma, prosedur alternative, data dalam bentuk tabel, kutipan dari dokumen lain, dan informasi relevan lainnya.

2.3 Dokumentasi Pengujian

Pengujian perangkat lunak merupakan sederetan langkah yang digunakan untuk melakukan pengujian atau pengecekan terhadap unit program ataupun sistem lengkap dari perangkat lunak untuk menjamin bahwa persyaratan sistem telah dipenuhi. Pengujian memastikan bahwa program tersebut telah berfungsi sebagaimana mestinya. Rencana, hasil serta prosedur pengujian harus didokumentasikan dalam suatu dokumen pengujian. Gambar 5 menunjukkan outline dari dokumen pengujian.

2.4 Dokumentasi Pengguna

Dokumentasi pengguna merupakan dokumen yang menyertai sebuah perangkat lunak yang berisi penjelasan secara detail tentang perangkat lunak tersebut. Dokumen pengguna menjelaskan setiap feature dari perangkat lunak dan menjelaskan bagaimana cara menggunakan setiap feature tersebut. Selain itu dokumen pengguna juga dapat memberikan penjelasan terhadap setiap masalah atau error yang terjadi dan bagaimana cara menanganinya. Dokumen pengguna dapat berupa dokumen cetak, elektronik, dokumen online yang mudah diakses ataupun gabungan dari semuanya. Dengan adanya dokumen pengguna ini, pengguna dapat dimudahkan dalam menggunakan perangkat lunak tersebut. IEEE telah mendefinisikan standar untuk dokumentasi pengguna. Pada standar tersebut, IEEE mendefinisikan komponen-komponen yang semestinya ada pada dokumentasi pengguna. Komponen yang disarankan oleh IEEE dapat dijadikan panduan untuk membuat dokumentasi pengguna.

3. KUALITAS DOKUMENTASI

Berdasarkan hasil survei yang dilakukan oleh Andrew Forward, software engineers mengungkapkan dokumen seperti apa yang dianggap berkualitas bagus, jelek dan sangat buruk [9].

1. Dokumen berkualitas bagus

- ☐ Arsitektur dan informasi dokumentasi lainnya selalu valid atau setidaknya menyediakan panduan sejarah yang dapat berguna untuk pemeliharaan perangkat lunak.
- ☐ Inline comments pada kode program cukup baik dalam memberikan informasi yang berguna untuk pemeliharaan perangkat lunak.

2. Dokumen berkualitas jelek

- ☐ Dokumentasi untuk semua jenis sering sekali tidak diperbaharui (out of date)
- ☐ Sistem mempunyai terlalu banyak dokumentasi
- ☐ Penulisan dokumentasi yang buruk
- ☐ Pengguna kesulitan menemukan isi yang berguna dalam dokumentasi
- ☐ Pembuatan dokumentasi yang memakan waktu yang tidak sebanding dengan keuntungan dari dokumentasi tersebut

3. Dokumen berkualitas sangat buruk

- ☐ Sebuah dokumentasi yang informasinya tidak dapat dipercaya

Secara umum dokumentasi yang bagus yaitu dokumen yang ditulis dengan baik, mudah dibaca dan dimengerti serta memberikan informasi yang lengkap dan akurat. Walaupun pembuatan dokumen yang seperti ini mungkin akan menyita waktu yang lebih banyak, tetapi dengan dokumen yang baik akan sangat membantu baik itu pengembang maupun pengguna program tersebut.

Berdasarkan survei Andrew Forward [10] menunjukkan bahwa isi dokumen merupakan atribut dokumen yang paling penting dari sebuah dokumentasi perangkat lunak. Tiga atribut lainnya yang dianggap penting yaitu up-to-date, availability, use of examples. Atribut-atribut tersebut yang sangat menentukan kualitas suatu dokumen, walaupun atribut lainnya juga tidak kalah pentingnya.

4. ALAT BANTU

Ada banyak software tool yang dapat digunakan untuk membantu membuat dokumentasi perangkat lunak. Penggunaan tool dapat mempercepat dan mempermudah dalam pembuatan dokumentasi.

Teknologi lainnya yang digunakan pengembang berdasarkan survei tersebut yaitu ArgoUML, Visio, FrameMaker, AuthorIT, whiteboards dan digital cameras, JUnit dan XML editors. Word processors paling banyak digunakan karena merupakan tool yang gampang digunakan dan lebih fleksibel. Tool yang berguna lainnya yaitu software sejenis mindmap (freemind).

Software tersebut dapat membantu untuk pengembang dalam menuliskan dokumentasi terkait dengan pembuatan perangkat lunak.

Daftar Pustaka

- R. S. Pressman, Software Engineering: A Practitioner's Approach. New York: Mc Graw-Hill Companies, Inc, 1997.
- R. A.S and M. Shalahuddin, Modul Pembelajaran Rekayasa Perangkat Lunak (Terstruktur dan Berorientasi Objek). Bandung:Modula, 2011.
- A. Forward, "Software Documentation – Building and Maintaining Artefacts of Communication," University of Ottawa, 2002.
- I. Sommerville, "Software Documentation," 2011.
- D. L. Parnas, "Precise Documentation : The Key To Better Software," Joburg Centre for Software Engineering.
- R. Shujaat, "Types of Software Documentation," 2009. [Online]. Available: <http://rafia-shujaat.suite101.com/types-of-softwaredocumentation-a107716>. [Accessed: 23-Apr-2012].
- I. Sommerville, Software Engineering, 06 ed. London: Pearson Education, 2000.
- IEEE, IEEE Standard for Software User Documentation, IEEE-Std1063-2001. New York: Institute of Electrical and Electronics Engineers, 2001.
- T. C. Lethbridge, J. Singer, A. Forward, and D. Consulting, "How Software Engineers Use Documentation : The State of the Practice Documentation :," IEEE Computer Society, pp. 35-39, 2003.
- A. Forward, K. Edward, and T. C. Lethbridge, "Software Engineering Documentation Priorities : An Industrial Study," University of Ottawa, pp. 1-13, 2002.
- A. Forward, K. Edward, and T. C. Lethbridge, "The Relevance of Software Documentation , Tools and Technologies : A Survey," University of Ottawa, 2002.
- [M. D. Ernst, "Automated documentation inference to explain failed tests," 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 63-72, Nov. 2011.
- Najwaini, Effan and SN, Azhari, " Dokumentasi sebagai bagian dari perangkat lunak," Semantik 2012, ISBN 979-26-0255-0

